Packt>

异步图书
www.epubit.com.cn



使用无监督学习建立自动化的预测和分类模型

深度学习精要（基于R语言）

R Deep Learning Essentials

[美] Joshua F. Wiley 著

高蓉 译

中国工信出版集团　人民邮电出版社
POSTS & TELECOM PRESS



深度学习精要（基于R语言）

[美] Joshua F. Wiley 著
高蓉 译

人 民 邮 电 出 版 社
北 京

图书在版编目（C I P）数据

深度学习精要 : 基于R语言 / (美) 威利
(Joshua F. Wiley) 著 ; 高蓉译. -- 北京 : 人民邮电
出版社, 2017.9
ISBN 978-7-115-46415-6

Ⅰ. ①深… Ⅱ. ①威… ②高… Ⅲ. ①程序语言一程
序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2017)第183745号

版权声明

Copyright ©2016 Packt Publishing. First published in the English language under the title R Deep Learning Essentials.

All rights reserved.

本书由英国 Packt Publishing 公司授权人民邮电出版社出版。未经出版者书面许可，对本书的任何部分不得以任何方式或任何手段复制和传播。

版权所有，侵权必究。

◆ 著　　[美] Joshua F. Wiley
译　　高　蓉
责任编辑　陈冀康
责任印制　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000　1/16
印张：10.75
字数：207 千字　2017 年 9 月第 1 版
印数：1 – 2 400 册　2017 年 9 月河北第 1 次印刷

著作权合同登记号　图字：01-2016-7607 号

定价：49.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

内容提要

本书重点介绍如何将 R 语言和深度学习模型或深度神经网络结合起来，解决实际的应用需求。全书共 6 章，分别介绍了深度这习基础知识、训练预测模型、如何防止过拟合、识别异常数据、训练深度预测模型以及调节和优化模型等内容。

本书适合了解机器学习概念和 R 语言并想要使用 R 提供的包来探索深度学习应用的读者学习参考。

作者简介

Dr. Joshua F. Wiley 是莫纳什大学的讲师，也是统计咨询公司 Elkhart 集团有限公司的资深合伙人。他从位于洛杉矶的加利福尼亚大学获得了博士学位。他的研究集中于使用高级数量方法来理解社会、心理，以及与心理和生理健康有关的生理过程之间的复杂的相互影响。在统计和社会科学方面，Joshua 关注生物统计并且对可重复性研究以及数据和统计模型的图形显示非常有兴趣。通过在 Elkhart 集团有限公司的顾问工作以及他之前在 UCLA 统计顾问集团的工作，Joshua 已经帮助过各种各样的客户，从试验研究者到生物技术公司。他开发或者共同开发了许多 R 包，包括 `varian`，一个用来构建贝叶斯尺度位置结构方程模型的包，以及 `MplusAutomation`，一个将 R 链接到商业软件 `Mplus` 的热门 R 包。

我要感谢我的妻子和家人多年来的支持与鼓励，使我对工作始终抱有热忱。

审阅人简介

Vincenzo Lomonaco，1991 年出生于意大利的圣乔瓦尼-罗通多。他在巴西利卡塔度过了童年时代，在获得了科学学院文凭之后，他搬到了摩德纳。之后不到 3 年，他以优异的成绩从计算机科学专业毕业。由于受到博洛尼亚盛名和研究活动的吸引，他决定在那里开始计算机硕士的学习。2015 年，他以优异成绩毕业，毕业论文是《用于计算机视觉的深度学习：卷积神经网络和分层时间记忆在目标识别任务中的比较》。目前，他是博洛尼亚大学的博士研究生，研究深度学习和生物启发模式识别。

前言

本书主要介绍如何在 R 编程语言和环境当中训练并使用深度学习模型或深度神经网络。本书无意于提供有关深度神经网络的深入的理论覆盖，但它将给你足够的理论背景，帮助你理解深度神经网络的基础、应用以及结果的解释。本书还将提供一些包和函数，用来训练深度神经网络，优化它们的超参数来提升模型的准确度、生成预测或者建立模型的其他应用。为了着手处理现实生活中的例子和应用，本书将提供关于深度学习要领的易于阅读的全面介绍。

本书的内容

第 1 章“深度学习入门”，展示如何创建 R 和 H2O 包并安装在计算机或服务器上，内容涉及所有和深度学习有关的基本概念。

第 2 章“训练预测模型”，涉及如何训练一个浅层无监督的神经网络预测模型。

第 3 章“防止过拟合”，解释了可用于防止模型过拟合数据的不同方法，为了

提升泛化能力，叫作无监督数据上的正则化。

第 4 章“识别异常数据”，涉及识别异常数据，比如欺诈活动或者离群点，和如何执行无监督深度学习。

第 5 章“训练深度预测模型”，展示了如何训练深度神经网络来解决预测或分类问题，比如图像识别。

第 6 章“调节和优化模型”，解释了如何调整模型的调节参数来提升并优化深度学习模型的准确度和性能。

附录即文献包含了本书所有引用的参考书目。

预备知识

使用这本书，你不需要掌握太多的知识。你所需要的软件的主要部分是 R，它是开源的，可以在 Windows、Mac OS 和多种 Linux 上运行。你还需要最新版本的 Java。当你安装好了 R 和 Java，你还需要安装一些 R 包，所有这些 R 包都可以在主流的操作系统上工作。

或许，更具有挑战性的要求是，对于真正的深度学习应用，哪怕是探索非常小的例子，都要求有现代的硬件。在本书中，笔者主要使用的台式机，配置为 2.50 GHz 的 Intel Xeon E5-2670 v2（10 个物理核，20 个逻辑核），32GB 的内存和三星 850 PRO 512GB SSD。你不一定需要一个相同的系统，但是笔者发现在 16GB 内存、双核 i7 处理器的笔记本电脑上运行某些例子是很耗费时间的。

目标读者

本书适合那些有追求的数据科学家，他们熟知机器学习概念和 R，并且正在使

用 R 提供的包来探索深度学习范式。你最好对 R 语言有一个基本的理解而且对统计算法和机器学习技术运用自如，但你并不需要精通深度学习的概念。

排版约定

在本书中，你会发现许多文本样式，它们区分了不同类型的信息。这里是一些有关这些样式的例子，解释了它们的含义。

文本中的代码、数据库表名称、文件夹名称、文件名称、文件扩展名、路径名称、虚拟 URLs、用户输入以及推特用户定位显示如下所示。

“当然，我们无法真正使用 `library()` 函数，除非我们安装了这些包。”

代码块的设置如下所示。

```
## uncomment to install the checkpoint package
## install.packages("checkpoint")
library(checkpoint)

checkpoint("2016-02-20", R.version = "3.2.3")
```

当我们希望把你的注意力吸引到代码块的一个特别部分的时候，我们会将有关的行或项目设置成粗体，如下所示。

```
performance.outsample[,-4]
  Size Maxit Shuffle Accuracy AccuracyLower AccuracyUpper
1   40    60   FALSE     0.93          0.92          0.94
2   20   100   FALSE     0.92          0.91          0.93
3   20   100    TRUE     0.92          0.91          0.93
4   50   100   FALSE     0.91          0.90          0.92
5   50   100   FALSE     0.92          0.91          0.93
```

命令行的输入和输出形式如下所示。

```
h2oiris <- as.h2o(
  droplevels(iris[1:100, ]))
```

新术语或者重要的词汇用粗体显示。

> 警告或者重要的注释会出现在类似这样的方框中。

> 提示或技巧会类似这样出现。

读者反馈

我们始终欢迎来自读者的反馈，请让我们知道你对这本书的看法——喜欢哪些内容，不喜欢哪些内容。读者的反馈对我们来说十分重要，这样我们才能出版读者最需要的书。

常规的反馈请通过电子邮件发送到 feedback@packtpub.com，在邮件中请注明书名。

如果你是某方面的专家，有兴趣写书，或者为某一本书投稿，请阅读我们的作者指南，地址是 www.packtpub.com/authors。

客户支持

现在你已经是一本 Packt 图书的光荣的拥有者，为了让你的付出得到最大的回报，我们还将为你提供其他方面的服务。

下载示例代码

你可以登录 http://www.packtpub.com 的账户，下载本书的示例代码文件。如果你是从其他地方购买的本书，可以访问 http://www.packtpub.com/support，注册之后，我们会为你发送一封附有文件的电子邮件。

你可以根据下面的步骤下载代码文件：

（1）使用你的电子邮件地址和密码在我们的网站登录或者注册；

（2）将鼠标指针悬停在顶部的 SUPPORT 选项卡上；

（3）单击 Code Downloads & Errata；

（4）在搜索框中输入该书的名称；

（5）选择你要找的书来下载代码文件；

（6）在下拉菜单中选择你从何处购买了这本书；

（7）单击 Code Download（代码下载）。

如果下载了文件，请你确保使用下列软件的最新版本来解压缩或者提取文件夹。

- Windows：WinRAR / 7-Zip
- Mac：Zipeg / iZip / UnRarX
- Linux：7-Zip / PeaZip

下载本书的彩色图像

我们还为你提供了本书中所用的屏幕截图/示意图彩色图像的 PDF 文件。彩色图像能更好地帮助你理解输出中的变化。你可以从 https://www.packtpub.com/sites/default/files/downloads/RDeepLearningEssentials_ColorImages.pdf. 下载这些文件。

勘误

尽管我们会尽全力确保书中内容的准确性，但是错误仍然在所难免。如果你在我们的某本书中发现了错误——文字错误或者代码错误，而且愿意为我们报告这些错误，我们将感激不尽。这样不仅可以消除其他读者的挫败感，而且能帮助我们改进这本书的后续版本。如果你发现了任何的错误，可以访问 http://www.packtpub.com/submit-errata 来提交，选择你的书，单击 Errata Submission Form（勘误表提交表单），输入勘误详情。勘误通过验证之后，你提交的内容会被接受而且勘误会上传到我们的网站，或者添加到这本书勘误部分现有的勘误列表中。

如果你想查看之前提交的勘误，请访问 https://www.packtpub.com/books/content/support，在搜索栏中输入书名，所查询的信息会出现在勘误部分。

盗版举报

对所有的媒体来说，在互联网上剽窃版权材料都是一个棘手的问题。Packt 很重视保护我们的版权和许可。如果你在互联网上发现我们产品的任何形式的非法复制品，请立即告知我们的网址和网站名称，这样我们可以采取补救措施。

如果你发现可疑的盗版材料，请通过 copyright@packtpub.com 联系我们。

你的举报有助于保护作者的权益以及我们为你提供有价值内容的能力，我们对此深表感谢。

问题解答

如果你对本书的任何方面有疑问，可以通过 questions@packtpub.com 联系我们，我们会尽力解决问题。

目录

第 1 章 深度学习入门

本章讨论深度学习，这是一种强大的多层架构，可以用于模式识别、信号检测以及分类或预测等多个领域。深度学习并不新鲜，但在过去十年它获得了极高的关注，这部分归功于计算能力的不断发展和训练模型不断涌现出更有效的新方法，也源于可使用的数据量不断增加。在本章中，我们将学习深度学习是什么，训练这种模型有哪些 R 包，如何建立分析系统以及如何连接 R 和 H2O。在随后的章节，我们会将 H2O 用于许多案例，这些案例探讨如何真正训练和使用一个深度学习模型。

本章包括以下内容。

- 什么是深度学习？
- 使用 R 包来训练深度学习模型，如深度信念网络或深度神经网络。
- 连接 R 和 H2O，深度学习使用 H2O。

1.1 什么是深度学习

为了理解深度学习是什么，最简单的方式也许是首先理解常规机器学习是什么。一般来说，机器学习主要用于开发和使用那些从原始数据中学习、总结出来的

用于进行预测的算法。预测是个非常笼统的术语。例如，机器学习中的预测可以包括预测某位消费者将会在一家给定的公司花费是多少，或者预测一笔特殊的信用卡消费中是否存在欺诈。预测也包括更一般的模式识别，如给定的图片显示了什么字母，或者这张照片中是否有马、狗、人、脸、建筑等。深度学习是机器学习的一个分支，其中的深度（多层）架构用于映射输入或观测特征与输出之间的联系。这种深度架构使得深度学习特别适合处理含有大量变量的问题，同时可以把深度学习生成的特征当作学习算法整体的一部分，而不是把特征生成当作一个单独步骤。现已证明，深度学习在图像识别（包括笔迹以及图片或者物体的识别）和自然语言处理（如语音识别）领域非常有效。

现在已有许多类型的机器学习算法。在本书中，我们主要讨论神经网络，因为它在深度学习中非常流行。但是，这种侧重并不意味着这就是用于机器学习甚至深度学习的唯一技术，也不是说其他的技术没有价值或者不适合，技术的选择取决于具体的任务。我们将在 1.2 节从概念上更深入地讨论神经网络和深度神经网络是什么。

1.2 神经网络的概念综述

神经网络正如其名所示，命名的灵感源于身体中的神经过程和神经元。神经网络包括一系列的神经元，或者叫作节点，它们彼此连结并处理输入。神经元之间的连结经过加权处理，权重取决于从数据中学习、总结出的使用函数。一组神经元的激活和权重（从数据中自适应地学习）可以提供给其他的神经元，其中一些最终神经元的激活就是预测。

为了将这个过程刻画得更具体，我们借助于一个来自人类视觉感知的例子来加强理解。祖母细胞这个术语用于指这样一个概念，在大脑的某个地方有一个细胞或者神经元，它专门只对某个复杂的特定对象有反应，如我们的祖母。这种特性需要数千个细胞来代表我们遇到的每个独特实体或对象。相反的观点是，人们通过汇集更多的基本片断来建立复杂的表达方式从而形成了视觉感知。图 1-1 是一张正方形

的图片。

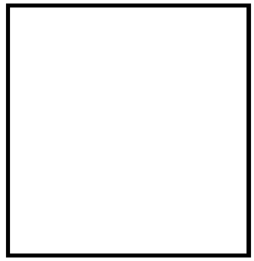


图 1-1

我们的视觉系统中有神经元而没有细胞，只要看到完整的正方形，神经元就会激活。我们的细胞可以识别图 1-2 所示的水平线和垂直线。

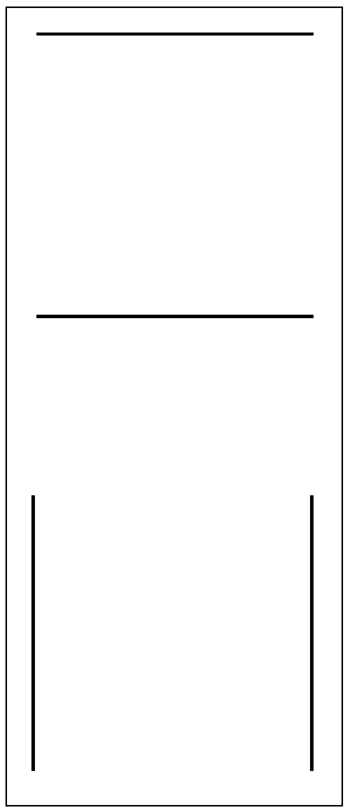


图 1-2

在这种假设情况中，有两个神经元，其中一个一旦感知到水平线就会被激活，另一个一旦感知到垂直线就会被激活。最后，一旦两个低阶的神经元同时激活，一个更高级的过程就会识别出它是一个正方形。

神经网络共享了一些相同的概念，输入经过第一层神经元的处理成为可以到达其他层的神经元。神经网络有时可以表示为图模型。如图 1-3 所示，输入是表示为方形的数据。它们可能是图像中的像素，或者是声音的不同方面，或者是其他的东西。接下来的一层隐藏神经元由诸如水平线、垂直线或者曲线这些基本特征的神经元组成。最后，输出是这样一个神经元，它通过同时激活两个隐藏的神经元而激活自身。在本书中，已观测的数据或特征为方形，而未观测的或隐藏层为圆形。

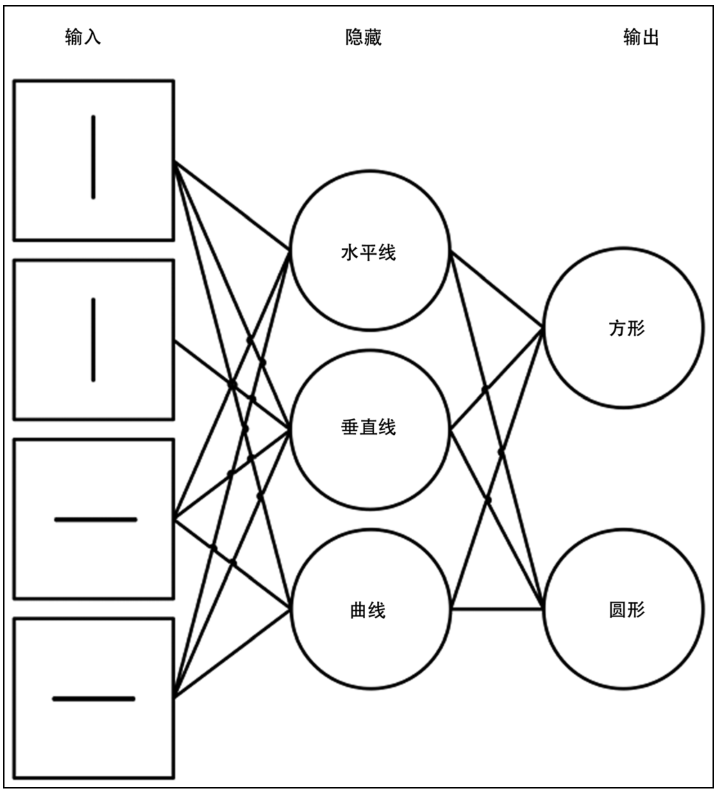


图 1-3

神经网络用来指示一类广泛的模型和算法。类似于其他统计技术的一个基础扩

展，神经网络基于一些观测数据的集合生成隐藏的神经元。但是，创建隐藏神经元的权重是从数据中学习得到的，而非通过扩展形式的选择而得到。神经网络可以包含多种激活函数，它们是加权原始数据输入的变换，用以创建隐藏神经元。经常选择的激活函数是 sigmoid 函数：$\sigma(x)=\frac{1}{1+e^{-x}}$ 以及双曲正切函数 $f(x)=\tanh(x)$。最后，因为径向基函数是有效的函数逼近，所以有时也会用到它们。尽管径向基函数种类很多，但高斯形式很常用：$f(x)=\exp\left(-\frac{\|x-c\|^2}{2\sigma^2}\right)$。

在一个类似于如图 1-3 所示的浅层神经网络中，只有一个隐藏层，从隐藏单元到输出，在本质上是一个标准的回归或分类问题。隐藏单元可以表示为 h，输出可以表示为 Y。不同的输出可以通过角标 $i=1,\ldots,k$ 来表示，代表不同的可能分类，例如（在我们的例子里）圆形或者方形。权重是从每个隐藏单元到每个输出的路径，对第 i 个的输出通过(w_i)表示。如创建隐藏层的权重，这些权重也是从数据中学习得到的。分类会经常使用一种最终变换，softmax 函数，即 $Y_i=\frac{e^{w_i^T h}}{\sum_i^k e^{w_i^T h}}$，这确保了估计为正（使用指数函数），并且任何给出类的概率和为 1。线性回归经常使用恒等（identity）函数，它返回输入值。关于为什么每个隐藏单元和输出之间存在路径，而每个输入和隐藏单元之间也存在路径，这很可能引起混乱。这通常用来表示允许任何先验关系存在。权重必须从数据中学习得到，权重为零或接近零基本上等同于放弃不必要的关系。

这里我们只介绍了神经网络概念与实践方面的初级内容。如果想要更深入地了解神经网络，我们可以参见 Hastie, T., Tibshirani, R., and Friedman, J. (2009)的第 11 章（在 `http://statweb. stanford.edu/~tibs/ElemStatLearn/` 有它的免费版本）、Murphy, K. P. （2012）的第 6 章和 Bishop, C. M.（2006）的第 5 章。下面，我们简要介绍深度神经网络。

1.3 深度神经网络

可能最简单的深度神经网络（deep neural network，DNN）的定义是，这是一种有多个隐藏层的神经网络，虽然这个定义的信息量并不是最大的。这种深度架构是神经网络的一种相对简单的概念扩展，它有效地推进了模型的能力和训练模型的新挑战。

多个隐藏层的使用允许模型实现由简到繁的更复杂的建立机制。在讨论神经网络时，我们考虑输出的是圆形还是方形。在深度神经网络中，多个圆形或方形可以组合形成其他更高级的形状。我们可以考虑模型架构的复杂性的两个方面。一个方面是它的宽窄如何——即给定的一层中有多少个神经元。第二个方面是它的深度如何，或者说神经元有多少层。对于确实具有这种深度架构的数据来说，相比于神经网络，深度神经网络可以使用更少的参数获得更精确的拟合结果，因为层数（每一层有较少的神经元）越多，意味着表现越有效而且越精确。比如，较浅的神经网络与深度神经网络相比，必须表示出每个独特的对象才能达到相等的精度，但它无法将基本部分整合出更高级的形状。再次考虑图像中的模式识别，如果我们想使用文本识别训练模型，原始数据可能是来自图像中的像素。我们可以训练第一层神经元来捕捉字母表中不同的字母，而另一层可以识别单词的字母集合。这样做优点是第二层无需直接从充满噪音的复杂像素中学习。相反，一个较浅的架构则可能需要多得多的参数，因为每个隐藏的神经元都需要具备从图像中的像素映射到复杂单词的能力，而许多单词都可能重叠，因此会造成模型中的冗余。

训练深度神经网络的挑战之一是如何通过有效学习得出权重。这些模型通常比较复杂，而且容易陷入局部最优化，因此最优化问题成为一种巨大挑战。2006 年出现了一个重大进展，它表明深度信念网络（Deep Belief Networks ，DBNs）可以一次训练一层［参见 Hinton, G. E., Osindero, S., and Teh, Y. W.（2006）］。DBN 是深度神经网络的一个类型，它含有多个隐藏层，层与层之间（而不是在内部）有连结（就是说，第一层的一个神经元可以和第二层的一个神经元连结，但不能和第一层的其

他神经元连结）。这个定义在本质上与受限玻尔兹曼机（Restricted Boltzmann Machine，RBM）相同，除了 RBM 通常具有一个输入层和一个隐藏层，受限玻尔兹曼机如图 1-4 所示。

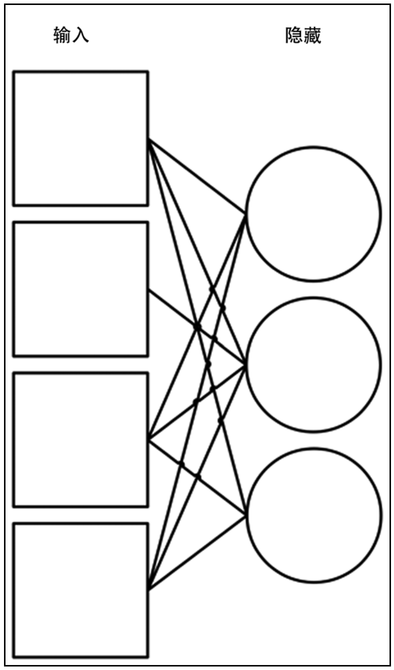


图 1-4

在一层之内没有连接的限制很有价值，因为它允许使用更快的训练算法，比如对比散度（contrastive divergence）算法。多个 RBM 叠加在一起，就能形成一个 DBN。本质上来说，DBN 可以被训练为一系列 RBMs。第一个 RBM 层训练出来向隐藏神经元转换原始数据，再在第二个 RBM 层中作为新的输入进行训练，这个过程一直重复直到所有层都经过训练。

这种实现的好处是 DBNs 可以一次只训练一层，还不仅限于 DBNs。DBNs 有时也用于深度学习网络的预处理阶段。它允许使用相对快速的逐层贪婪训练提供良好的初始估计，再在深度神经网络中使用其他较慢的训练进行改进，比如反向传播方法。

到目前为止，我们主要关注前馈神经网络，其中一层神经元的结果向前传递到下一层。在结束本节之前，我们需要注意两种日益流行的深度神经网络类型。第一种是循环神经网络（Recurrent Neural Network，RNN），它的神经元彼此间发送反馈信号。这种反馈循环使得 RNN 在序列上表现良好。一个最近的 RNN 应用例子是自动生成钓鱼点击，例如“发廊故意隐藏的大秘密”或者“造访拉斯维加斯的十大理由：第六个一定震撼你！”。RNN 十分擅长处理工作，因为 RNN 可以从一些单词的巨大初始池中取样（甚至只是趋势性的搜索术语或者名称），再预测生成下一个单词的内容。这个过程会重复若干次，直至生成一个简短叙述——钓鱼点击。这个例子来自 Lars Eidnes 的一篇博客（`http://larseidnes.com/2015/10/13/auto-generating-clickbait-with-recurrentneural-networks/`）。第二种类型是卷积神经网络（Convolutional Neural Network，CNN）。CNN 最常用于图像识别。它的工作原理是每个神经元都对图像的重叠部分作出响应。这种类型的优点是它们要求的预处理相对最小，但权重共享（比如说，穿过图像的重叠区域）仍然不需要太多的参数。图像常常并不一致，因此 CNN 对图像特别有价值。例如，10 个人拍摄同一张桌子的照片，有些人比较近，有些人比较远，有些人刚好在桌子旁边，位置差异导致了本质相同的图像在高度、宽度以及围绕焦点对象拍摄的照片数量的差异。

对神经网络来说，上述叙述只是规定了什么是神经网络以及一些相关应用案例的最简短的概述。概述我们可以参见 Schmidhuber, J.（2015）以及 Murphy，K. P.（2012）的第 28 章。

1.4 用于深度学习的 R 包

尽管 R 有大量用于机器学习的包，但用于神经网络和深度学习的包相对较少。在本节中，我们将讨论如何安装和设置所有需要的 R 包，以便使用神经网络和深度学习。

就使用 R 做数据分析来说，一个好的综合开发环境（integrated development

environment，IDE）很有帮助。我们选择使用 Emacs，它与 Emacs Speaks Statistics（ESS）相结合是一种强大的文本编辑器，有助于 Emacs 与 R 结合在一起很好地工作。方便的启动和运行方式是使用一种特定的 Emacs 修正发行版本，这个版本专门用来结合 R 一起工作和用于统计，而且效果很好。它由 Vincent Goulet 创建并维护，可以在 http://vgoulet.act. ulaval.ca/en/emacs/中免费获取。另一种流行的 R 的 IDE 是 Rstudio（https://www. rstudio.com/）。Emacs 和 Rstudio 的一个共同优点是它们在所有主要的平台上（Windows、Mac 和 Linux）都可用，因此即使我们更换电脑，IDE 经验仍然可以使用。

1.5 建立可重复的结果

数据科学软件正处于高速发展的阶段。尽管这对技术进步来说相当美妙，但对需要重复别人结果的人来说则构成了挑战。即使是我们自己的代码，几个月后再使用时也有可能无法工作。解决这个问题有一种方法，就是记录使用的软件版本并确保其快照可用。本书会使用 Revolution Analytics 公司提供的 R 包 checkpoint，它先连接上它们公司的服务器再运行，服务器提供了 R 综合档案网络（Comprehensive R Archive Network，CRAN）的每日快照（checkpoints）。想要了解这个过程的更多内容，我们可以在 https://cran.r-project.org/web/packages/checkpoint/vignettes/checkpoint.html 阅读此包的在线小文章（vignette）。

本书使用了绰号为“木质圣诞树”（Wooden Christmas-Tree）的 R3.2.3 版本，操作系统为 Windows 10 Professional x64。这个版本在本书写作期间是最新的。随着更新版本的发布，CRAN 会以二元形式（将来在 https://cran.r-project.org/bin/ windows/ base/old/）和源打包文件的形式（https://cran.r-project.org/ src/base/R-3/）保留旧版本的备份，用于在任何操作系统上编译源文件。

H2O 是使用深度学习的主要 R 包之一，我们需要安装 Java。本书使用了 64 位

的 Java SE Development Kit 8 update 66。我们可以从 `http://www.oracle.com/technetwork/java/javase/`下载适合自己的操作系统的 Java。

完成这些步骤后，我们就准备开始。为了使用 checkpoint 包，我们需要把用于同一个项目的 R 脚本放入同一个文件夹。安装使用 checkpoint 包的过程有些迂回。通过检查对 library()和 require()函数的调用，checkpoint 包的工作是扫描项目目录中的 R 脚本来看看载入了什么 R 包（因此它需要安装）。当然，除非这些包已经安装，我们实际不能使用 library()函数。

我们在项目目录下创建一个 R 脚本 checkpoint.R，使用代码如下所示。

```
## uncomment to install the checkpoint package
## install.packages("checkpoint")
library(checkpoint)

checkpoint("2016-02-20", R.version = "3.2.3")
```

一旦创建了 R 脚本，就可以取消注释并且运行代码来安装 checkpoint 包。这样我们只需要做一次，因此完成时最好再给代码添上注释，这样它不会在我们每次运行文件时都重新安装。而我们需要对深度学习项目创建 R 环境，这是我们每次都会运行的文件。本书的检查点是 2016 年 2 月 20 日，我们使用的 R 版本是 3.2.3。接下来，我们加入 library()调用一些需要的包，这些可以通过在 checkpoint.R 脚本中加入如下所示代码得到（但要注意，这些还没有运行！）。

```
## Chapter 1 ##

## Tools
library(RCurl)
library(jsonlite)
library(caret)
library(e1071)

## basic stats packages
```

```
library(statmod)
library(MASS)
```

提示：

下载示例代码

我们可以直接从 http://www.packtpub.com 下载所有已购买的 Packt 图书的示例代码文件。如果是从其他地方购买了本书，我们可以访问 http://www.packtpub.com/support 并注册，会通过电子邮件把文件发送给我们。

下载代码文件的步骤如下：

（1）登录网站；

（2）把鼠标按键悬停在顶部的 SUPPORT 标签上；

（3）单击“Code Downloads & Errata”；

（4）在搜索框输入书名；

（5）选择你想要下载代码文件的图书；

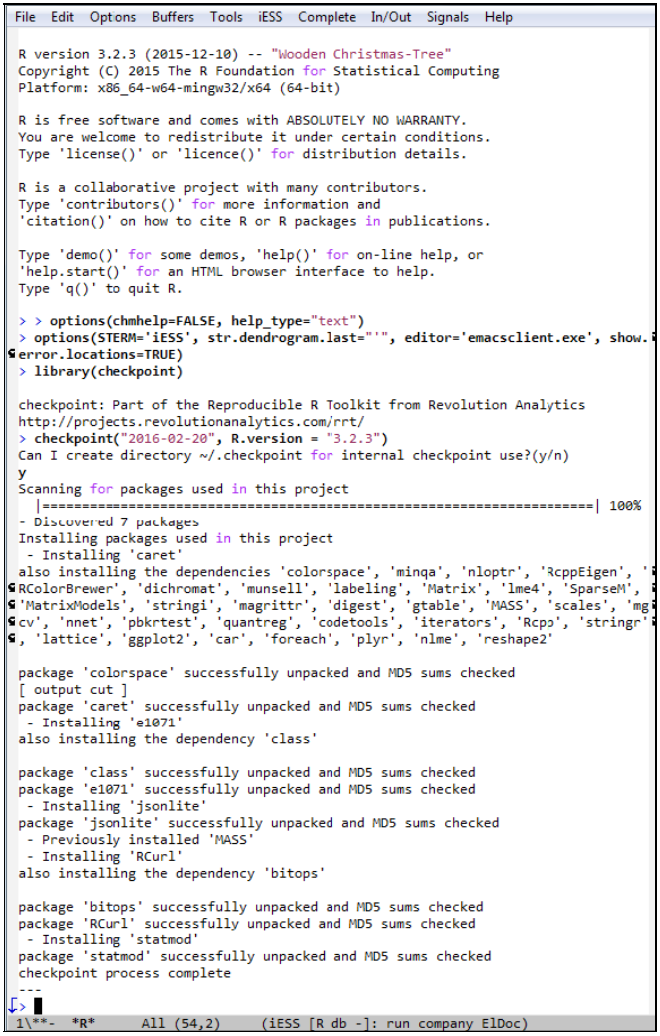
（6）在下拉菜单选择我们从何处购买了本书；

（7）单击“Code Download”。

文件下载后，确保我们使用以下软件的最新版本解压或提取文件夹：

- 对于 Windows 是 WinRAR / 7-Zip;
- 对于 Mac 是 Zipeg / iZip / UnRarX;
- 对于 Linux 是 7-Zip / PeaZip。

一旦加入代码、保存文件，任何改变都会写入磁盘，接下来我们运行前两行代码来载入 checkpoint 包并调用 checkpoint()，结果如图 1-5 所示。



```
File  Edit  Options  Buffers  Tools  iESS  Complete  In/Out  Signals  Help

R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> > options(chmhelp=FALSE, help_type="text")
> options(STERM='iESS', str.dendrogram.last="'", editor='emacsclient.exe', show.
error.locations=TRUE)
> library(checkpoint)

checkpoint: Part of the Reproducible R Toolkit from Revolution Analytics
http://projects.revolutionanalytics.com/rrt/
> checkpoint("2016-02-20", R.version = "3.2.3")
Can I create directory ~/.checkpoint for internal checkpoint use?(y/n)
y
Scanning for packages used in this project
  |====================================================================| 100%
- Discovered 7 packages
Installing packages used in this project
 - Installing 'caret'
also installing the dependencies 'colorspace', 'minqa', 'nloptr', 'RcppEigen', '
RColorBrewer', 'dichromat', 'munsell', 'labeling', 'Matrix', 'lme4', 'SparseM',
'MatrixModels', 'stringi', 'magrittr', 'digest', 'gtable', 'MASS', 'scales', 'mg
cv', 'nnet', 'pbkrtest', 'quantreg', 'codetools', 'iterators', 'Rcpp', 'stringr'
, 'lattice', 'ggplot2', 'car', 'foreach', 'plyr', 'nlme', 'reshape2'

package 'colorspace' successfully unpacked and MD5 sums checked
[ output cut ]
package 'caret' successfully unpacked and MD5 sums checked
 - Installing 'e1071'
also installing the dependency 'class'

package 'class' successfully unpacked and MD5 sums checked
package 'e1071' successfully unpacked and MD5 sums checked
 - Installing 'jsonlite'
package 'jsonlite' successfully unpacked and MD5 sums checked
 - Previously installed 'MASS'
 - Installing 'RCurl'
also installing the dependency 'bitops'

package 'bitops' successfully unpacked and MD5 sums checked
package 'RCurl' successfully unpacked and MD5 sums checked
 - Installing 'statmod'
package 'statmod' successfully unpacked and MD5 sums checked
checkpoint process complete
---
> 
 1\**-  *R*      All (54,2)     (iESS [R db -]: run company ElDoc)
```

图 1-5

checkpoint 包需要创建一个路径来存储使用包的具体版本，接着找到所有的包并安装。接下来展示如何准备用于深度学习的一些特定 R 包。

1.5.1　神经网络

在 R 中，有几个包可以拟合基本的神经网络。nnet 包值得推荐，可以拟合有一

个隐藏层的前馈神经网络，如图 1-3 所示。如果需要 nnet 包更多的细节，我们可以参见 Venables, W. N. and Ripley, B. D.（2002）。neuralnet 包也可以拟合有一个隐藏层的浅层神经网络，但它可以使用后向传播算法训练它们并允许自定义误差和神经元激活函数。最后，我们来讨论 RSNNS 包，它是斯图加特神经网络仿真器（Stuttgart Neural Network Simulator，SNNS）的一个 R 封装。最初，SNNS 用 C 编写，但后来移植到了 C++上。RSNNS 允许很多类型的模型在 R 中拟合。常见的模型使用方便的封装就可以训练，提供了常见的模型。但是 RSNN 包基于 SNN 的内容提供了许多模型的组成部分，因此它可以训练类型广泛的模型。如果需要 RSNNS 包更多的细节，我们可以参见 Bergmeir, C.和 Benítez, J. M.（2012）。在本书第 2 章中，我们将看到使用这些模型的例子。现在，我们可以将下列代码添加并保存到 `checkpoint.R` 脚本中，然后安装这些包。保存非常重要！因为如果 R 脚本的改变没有写到磁盘上，那么 `checkpoint()`函数就不会有改变，也找不到新包，更安装不了。

```
## neural networks
library(nnet)
library(neuralnet)
library(RSNNS)
```

现在，如果我们重运行 `checkpoint()`并成功，R 会告诉我们它发现了 8 个包，而且它安装了 nnet、neuralnet、RSNNS 以及 RSNNS 必需的 Rcpp 包。

1.5.2 deepnet 包

deepnet 包为 R 中的深度学习提供了许多工具。特别是它可以训练 RBM 并使用这些 RBM 作为 DBN 的一部分来生成初始值，进而训练深度神经网络。deepnet 包也接受不同的激活函数，同时使用丢弃来正则化。为了安装这个包，我们遵从之前使用的相同过程，为 `checkpoint.R` 添加代码，保存，再重新运行 `checkpoint()`函数。

```
## deep learning
library(deepnet)
```

1.5.3　darch 包

darch 包基于 George Hinton 开发的 Matlab 代码，它代表深度学习架构。它可以联合彼此相关的大量选项，来训练 RBM 和 DBM。darch 包有一个缺陷，因为它是一种纯粹的 R 实现，所以模型训练往往很慢。为了安装这个包，我们遵从之前使用的相同过程，为 `checkpoint.R` 添加代码，保存，再重新运行 `checkpoint()` 函数。

```
## deep learning
library(darch)
```

1.5.4　H2O 包

H2O 包提供了一个到 H2O 软件的接口。H2O 用 Java 编写，又快又可扩展。它不仅提供深度学习的功能，还提供许多其他流行的机器学习的算法和模型，另外，模型结果可以存储为纯粹的 Java 代码，可以进行快速评分、推进部署模型，从而解决真实世界的问题。为了安装这个包，我们遵从之前使用的相同过程，为 checkpoint.R 添加代码，保存，再重新运行 checkpoint()函数。

```
## deep learning
library(h2o)
```

1.6　连接 R 和 H2O

因为 H2O 是一个基于 Java 的 R 封装软件，为了与 R 连接，我们必须初始化一个 H2O 实例并把它和 R 连结起来，把数据和模型命令连结或传递给这个实例。在本节，我们将展示如何准备好去训练使用 H2O 的模型。

1.6.1　初始化 H2O

为了初始化一个 H2O 集群，我们使用 h2o.init()函数。集群的初始化也将建立一个轻量级的 Web 服务器，它可以通过本地网页与软件进行交互。h2o.init()函数通常都有合理的缺省值，但我们可以自定义它的许多方面，尤其好的是，自定义使用的核/线程的个数以及我们为它配置的内存大小，都可以在下列使用了 max_mem_size 和 nthreads 参数的代码中完成。在下面的代码中，我们使用两个线程和多达 3G 的内存来初始化 H2O 集群。代码运行之后，R 将给出日志文件的位置、Java 版本以及集群的细节。

```
cl <- h2o.init(
  max_mem_size = "3G",
  nthreads = 2)

H2O is not running yet, starting it now...

Note: In case of errors look at the following log files:
   C:\Users\jwile\AppData\Local\Temp\RtmpuelhZm/h2o_jwile_started_
from_r.out
   C:\Users\jwile\AppData\Local\Temp\RtmpuelhZm/h2o_jwile_started_
from_r.err

java version "1.8.0_66"
Java(TM) SE Runtime Environment (build 1.8.0_66-b18)
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b18, mixed mode)

.Successfully connected to http://127.0.0.1:54321/

R is connected to the H2O cluster:
    H2O cluster uptime:         1 seconds 735 milliseconds
    H2O cluster version:        3.6.0.8
    H2O cluster name:           H2O_started_from_R_jwile_ndx127
```

```
H2O cluster total nodes:    1
H2O cluster total memory:   2.67 GB
H2O cluster total cores:    4
H2O cluster allowed cores: 2
H2O cluster healthy:       TRUE
```

一旦这个集群完成初始化，我们可以使用 R 或本地主机(`127.0.0.1:54321`)提供的 Web 接口与它连接，如图 1-6 所示。

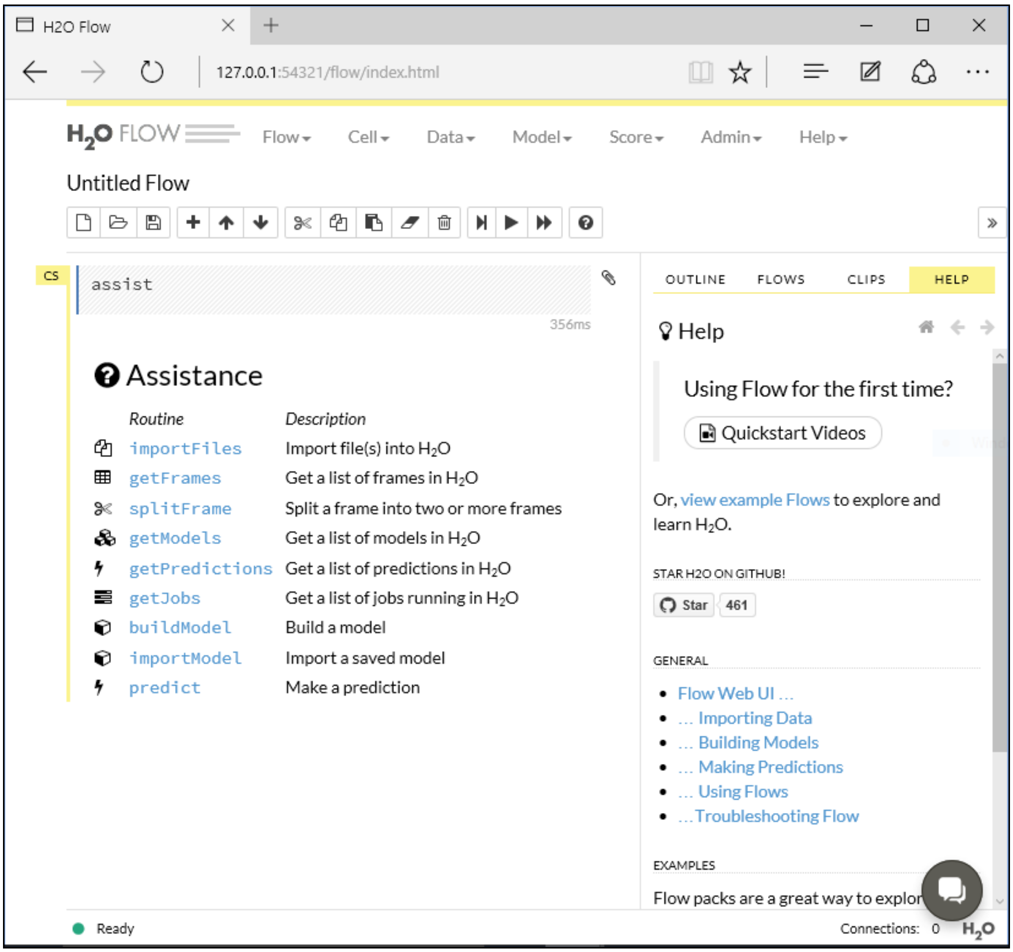


图 1-6

1.6.2　数据集连结到 H2O 集群

这里有几种方法可以将数据载入 H2O 集群。如果数据集已经下载到 R，我们只需要使用下面代码所示的 as.h2o()函数。

```
h2oiris <- as.h2o(
  droplevels(iris[1:100, ]))
```

我们可以通过键入 R 对象 h2oiris 来确认结果，这个对象只是一个包含了引用 H2O 数据的对象。当我们尝试打印它时，R API 会查询 H2O。

```
h2oiris
```

这里会返回如下输出。

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

[100 rows x 5 columns]
```

我们也能检查因子变量的水平，比如 Species 变量，代码如下所示。

```
h2o.levels(h2oiris, 5)
[1] setosa     versicolor
```

在真实世界的应用中，数据很可能已经在某处存在。与其把数据载入 R，不如导入 H2O（在 R 中对数据再创建一份不必要的副本代价昂贵），我们可以直接把数

据载入 H2O。首先，我们基于内置的 mtcars 数据集创建一个 CSV 文件，然后通知 H2O 实例使用 R 读数据，接着再次打印，数据显示如下所示。

```
write.csv(mtcars, file = "mtcars.csv")

h2omtcars <- h2o.importFile(
  path = "mtcars.csv")

h2omtcars

                   C1  mpg cyl disp  hp drat    wt  qsec vs am gear carb
1           Mazda RX4 21.0   6  160 110 3.90 2.620 16.46 0  1    4    4
2        azda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02 0  1    4    4
3          Datsun 710 22.8   4  108  93 3.85 2.320 18.61 1  1    4    1
4      Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44 1  0    3    1
5   Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02 0  0    3    2
6             Valiant 18.1   6  225 105 2.76 3.460 20.22 1  0    3    1
[32 rows x 12 columns]
```

最后，数据不必载入本地硬盘。在最后一个例子里，我们也能请求 H2O 从显示的 URL 读入数据。这个例子使用了 UCLA 统计咨询集团提供的数据集。

```
h2obin <- h2o.importFile(
  path = "http://www.ats.ucla.edu/stat/data/binary.csv")

h2obin
  admit gre  gpa  rank
1     0 380 3.61     3
2     1 660 3.67     3
3     1 800 4.00     1
4     1 640 3.19     4
5     0 520 2.93     4
6     1 760 3.00     2

[400 rows x 4 columns]
```

1.7 小结

本章简要介绍了神经网络和深度神经网络。深度神经网络使用了多个隐藏层，通过提供强大的无监督学习和特征提取部分，已成为机器学习中的一场革命，其中的特征提取部分可以独立使用，也可以作为有监督学习的集成部分。

这些模型有许多应用，而且还越来越多地在包括谷歌、微软和脸书这种大公司得到使用。深度学习的应用例子有图像识别（例如自动标注面孔，或者识别图像的关键词）、语音识别、文本翻译（例如从英语到西班牙语，反之亦然）。它甚至可以在文本识别上发挥作用，比如情感分析，它可以判断一个句子或一个段落大致上是正面的还是负面的，这对于评价一项产品或服务的看法很有用。设想我们可以获取与我们产品相关的任何评论和社交媒体，与一个月或者一年之前的情形对比分析，正面的讨论变得更多还是更少。

本章还展示了如何设置 R 以及需要安装的软件和包，本书采用的可重复方式匹配本书使用的版本。

在第 2 章中，我们开始训练神经网络并自己生成预测。

第 2 章 训练预测模型

本章通过实践例子来展示如何在 R 中建立并训练基本的神经网络。这些例子也强调了评价模型不同的调节参数的重要性从而找到最佳的参数集合。尽管评价多种调节参数有助于模型的性能提升，但是它也会引起过拟合的问题，这是本章讨论的第二个主题。本章结尾使用了数据分类的例子，活动数据来自于行走、上下楼梯、坐下、站立或躺倒时的智能手机。

本章包括下列主题。

- R 中的神经网络
- 过拟合数据的问题——结果解释
- 建立案例并运用神经网络方法

2.1 R 中的神经网络

为了在 R 中训练基本（即隐藏层数目为单个的“浅层的”）神经网络，我们将使用 nnet 包和 RSNNS 包［Bergmeir, C.和 Benítez, J. M.（2012）］。在第 1 章，这些包已经安装并通过了 2016 年 2 月 20 日的检查点，因此我们的结果完全可重复。尽

管可以与 nnet 包直接连接，我们还是通过 caret 包来使用它。caret 是“Classification and Regression Training”（分类与回归训练）的缩写。caret 包提供了标准化的接口与 R 中许多机器学习模型协作（Kuhn，2008；Kuhn 和 Johnson，2013），而且对验证和性能评估提供了许多有用的特征，我们会在本章和第 3 章中使用这些应用。

我们建立神经网络的第一个例子，这里使用了一个经典的分类问题——基于图片识别手写体数字。数据可以从 `https://www.kaggle.com/c/digit-recognizer` 下载，格式是易于使用的 CSV 格式，其中数据集的每一列（即特征），表示图像的一个像素。每张图像都经过标准化处理，转化成同样的大小，所以所有图像的像素个数都相同。第一列包含真实的数据标签，其余各列是黑暗像素的值，它用于分类。下载文件的名称分别为 `train.csv` 和 `test.csv`，和 R 脚本存放在同一个文件夹中，因此很容易读取。如果它们放入不同的文件夹，我们只需要相应改变路径。

2.1.1 建立神经网络

首先，我们先载入包，在我们需要载入的脚本中调用 `source()` 并对使用的版本设置检查点。然后，我们可以读取从 Kaggle 下载的数据并浏览它的样子。

```
source("checkpoint.R")
## output omitted
digits.train <- read.csv("train.csv")

dim(digits.train)

[1] 42000   785

head(colnames(digits.train), 4)

[1] "label"  "pixel0" "pixel1" "pixel2"

tail(colnames(digits.train), 4)
```

```
[1] "pixel780" "pixel781" "pixel782" "pixel783"

head(digits.train[, 1:4])

   label pixel0 pixel1 pixel2
1      1      0      0      0
2      0      0      0      0
3      1      0      0      0
4      4      0      0      0
5      0      0      0      0
6      0      0      0      0
```

我们将标签（数字 0～9）转换为因子，使 R 知道这是一个分类问题而不是回归问题。如果这个问题是真实的，那我们就尽量使用全部 42 000 个观测，但为了缩减运行时间，对这个例子我们只选择前 5 000 个观测来建立并训练神经网络。我们把数据分解成特征或预测子（digits.X）和输出（digits.Y），使用除标签之外所有的列作为这里的预测子。

```
## convert to factor
digits.train$label <- factor(digits.train$label, levels = 0:9)
i <- 1:5000
digits.X <- digits.train[i, -1]
digits.y <- digits.train[i, 1]
```

最后，在开始建立神经网络之前，我们需要先迅速检查一下数字的分布，这一点很重要。例如，一个数字很少出现，但同时我们还很关心这个数字的精确预测，那我们需要调整建模方法，来做到即使它很少出现，在评估性能时也会赋予它足够的权重。下面的代码片段创建了一个条形图，显示了每个数字标签的频数，如图 2-1 所示。它们的分布相当均匀，因此我们不需要对任何特别的数字增加权重或重要性。

```
barplot(table(digits.y))
```

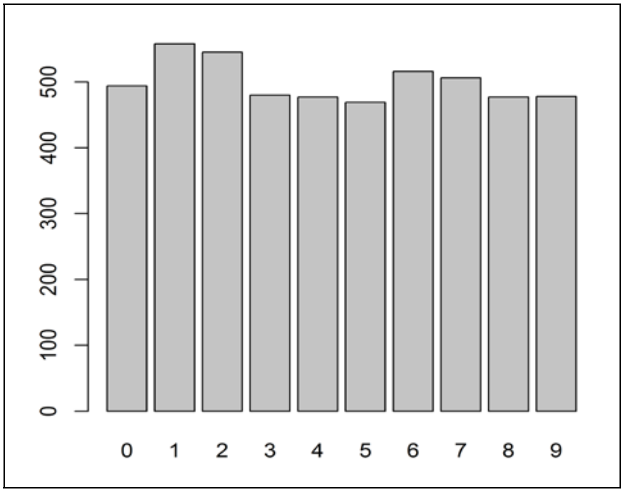


图 2-1

现在，我们通过 caret 包的封装来使用 nnet 包建立和训练第一个神经网络。首先，我们使用 `set.seed()`函数来指定特定的种子，使结果可以重复。精确的种子并不重要。重复相同的种子这种方法同样也运用于后面的例子，因为真正重要的是相同的模型使用相同的种子，而非不同的模型使用不同或相似的种子。`train()`函数首先取特征或预测子数据作为 x 参数，然后取输出变量作为 y 参数。`train()`可以处理许多模型，通过 method 参数来控制。虽然机器学习模型的许多方面都是自动学习的，但是某些参数还需要人为设置。它们随着使用方法而变。例如，在神经网络中，有一个参数是隐藏单元的个数。在需要尝试多种调节参数时，`train()`函数提供了一种容易的方法，将它作为一个指定的数据框传递给 tuneGrid 参数。对每个调节参数的集合，它都返回性能度量和最好的训练模型。我们的初始情况是，在模型中只有 5 个隐藏神经元以及一个适中的衰变率，有时它也称为“学习率”。学习率控制了每个迭代或步骤对当前权重的影响。另一个参数 `trControl` 控制了 `train()`的其他部分，当评估多种调节参数时，用于通知 caret 包如何证实并挑选出最佳的调节参数。

对这个例子，我们把训练控制的方法设置为 none，因为我们此时只使用了一组调节参数。最后，我们可以指定额外的命名参数，它们传递给真实的 `nnet()`函数

（或者无论指定的什么算法）。根据预测子的个数（784），我们把最大权重个数提高为 10 000 并指定最大迭代次数为 100。因为数据数量相对较小而且缺乏隐藏的神经元，运行这个模型的时间不会很长。

```
set.seed(1234)
digits.m1 <- train(x = digits.X, y = digits.y,
           method = "nnet",
           tuneGrid = expand.grid(
             .size = c(5),
             .decay = 0.1),
           trControl = trainControl(method = "none"),
           MaxNWts = 10000,
           maxit = 100)
```

predict()函数生成了数据的一组预测。我们在不指定任何新数据的情形下调用模型结果，结果它只在训练的相同数据上生成了预测。在计算和存储预测数字之后，我们可以检查它们的分布，如图 2-2 所示。即使在看到第一个模型的性能度量之前考虑到它的真实分布（图 2-1），很明显模型不是最优的。

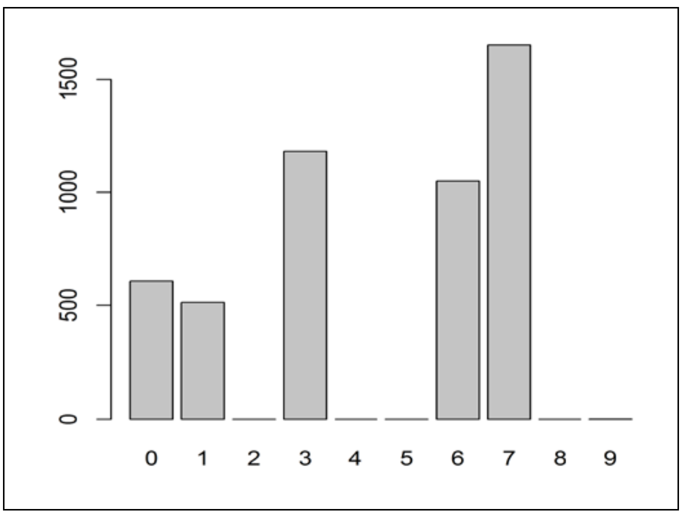


图 2-2

```
digits.yhat1 <- predict(digits.m1)
barplot(table(digits.yhat1))
```

分布的图形检查仅仅是预测的一个简单检查。模型性能的更正式评价是使用 caret 包的 `confusionMatrix()`函数。因为在 RSNNS 包中也有一个相同名称的函数，它们难以区分，所以我们使用专门的 caret::代码来告诉 R 使用的函数是哪个版本。输入仅仅是真实数字和预测数字之间的频率交叉表，其余的性能指标基于这个表来计算。

我们有多个数字，因此对性能输出我们给出三个主要部分。首先，我们给出真实的频率交叉表格。正确的预测位于对角线上，错误分类的各个频率位于对角线之外。接下来是总体统计量，它们是指所有类的模型表现。准确度仅仅指正确分类的实例比例，有一个 95%的置信区间，这个度量对较小的数据集特别有用，因为它的估计不确定性相当大。无信息率是指，不考虑任何信息而仅仅通过猜测来决定最频繁的类的准确度期望。在情形“1”中，它在 11.16%的时间中发生。P 值检验了观测准确度（44.3%）是否显著不同于无信息率（11.2%）。结果尽管在统计上是显著的，但对于数字分类来说并不是很有意义，我们本来期望会远远好于纯猜测的结果！最后，它对每个数字给出了个别的性能指标。这些指标都是基于数字对其他所有数字的比较而计算的，因此每个指标都是二元比较。表 2-1 包含了计算各种度量需要的所有信息，同时所有度量的公式也展示在这里。

表 2-1

	灵　敏　度	特　异　度
阳性预测值	True positive(TP)	False positive(FP)
阴性预测值	False negative(FN)	True negative(TN)

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Positive\ Predictive\ Value(PPV) = \frac{TP}{TP + FP}$$

$$Negative\ Predictive\ Value(NPV) = \frac{TN}{FN + TN}$$

$$Detection\ Rate = \frac{TP}{TP + FN + FP + TN}$$

$$Detection\ Prevalence = \frac{TP + FP}{TP + FN + FP + TN}$$

比如，数字“0”的灵敏度可以解释为，78.5%的数字“0”被捕捉或者正确地预测为数字“0”。数字“0”的特异度可以解释为，95.2%的预测为非数字“0”的实例并不是数字“0”。检出率仅仅是真阳性的百分比，而最后的检出预防度（detection prevalence）是预测为阳性的实例比例，不管它们是否真的为阳性。平衡准确度（balanced accuracy）是灵敏度和特异度的平均值。其余各列展示了其余各个数字的同样信息。

```
caret::confusionMatrix(xtabs(~digits.yhat1 + digits.y))

Confusion Matrix and Statistics

             digits.y
digits.yhat1   0   1   2   3   4   5   6   7   8   9
           0 388   2  40  41   7  75  23   4  23   2
           1   0 495   3   0   0   3   0   4   3   4
           2   0   0   0   0   0   0   0   0   0   0
           3  51  30  36 379   6 329   3  18 290  38
           4   0   0   0   0   0   0   0   0   0   0
           5   0   0   0   0   0   0   0   0   0   0
```

```
           6   44    5  304    9  131   29  484    9   16   19
           7   11   26  162   51  333   33    6  470  145  415
           8    0    0    0    0    0    0    0    0    0    0
           9    0    0    0    0    0    0    0    1    0    0
Overall Statistics

               Accuracy : 0.4432
                 95% CI : (0.4294, 0.4571)
    No Information Rate : 0.1116
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3805
Mcnemar's Test P-Value  : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.7854   0.8871    0.000   0.7896   0.0000
Specificity            0.9518   0.9962    1.000   0.8228   1.0000
Pos Pred Value         0.6413   0.9668      NaN   0.3212      NaN
Neg Pred Value         0.9759   0.9860    0.891   0.9736   0.9046
Prevalence             0.0988   0.1116    0.109   0.0960   0.0954
Detection Rate         0.0776   0.0990    0.000   0.0758   0.0000
Detection Prevalence 0.1210   0.1024    0.000   0.2360   0.0000
Balanced Accuracy      0.8686   0.9416    0.500   0.8062   0.5000
                     Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity            0.0000   0.9380   0.9289   0.0000   0.0000
Specificity            1.0000   0.8738   0.7370   1.0000   0.9998
Pos Pred Value            NaN   0.4610   0.2845      NaN   0.0000
Neg Pred Value         0.9062   0.9919   0.9892   0.9046   0.9044
Prevalence             0.0938   0.1032   0.1012   0.0954   0.0956
Detection Rate         0.0000   0.0968   0.0940   0.0000   0.0000
Detection Prevalence 0.0000   0.2100   0.3304   0.0000   0.0002
Balanced Accuracy      0.5000   0.9059   0.8329   0.5000   0.4999
```

现在，对于如何建立、训练和评价模型性能，我们已经有了一些基本理解。我们接下来尝试一些不同的模型，增加隐藏神经元的个数，这是提升模型性能的关键方法，其代价是模型复杂性的显著增加。回顾第 1 章，每个预测子或特征都和每个隐藏的神经元相连接，而且每个隐藏的神经元和都每个结果或输出相连接。每个增加的神经元有 784 个特征，再加上大量的参数，导致了运行时间更长。计算取决于我们的电脑性能，接下来要为完成这些模型等待一段时间。

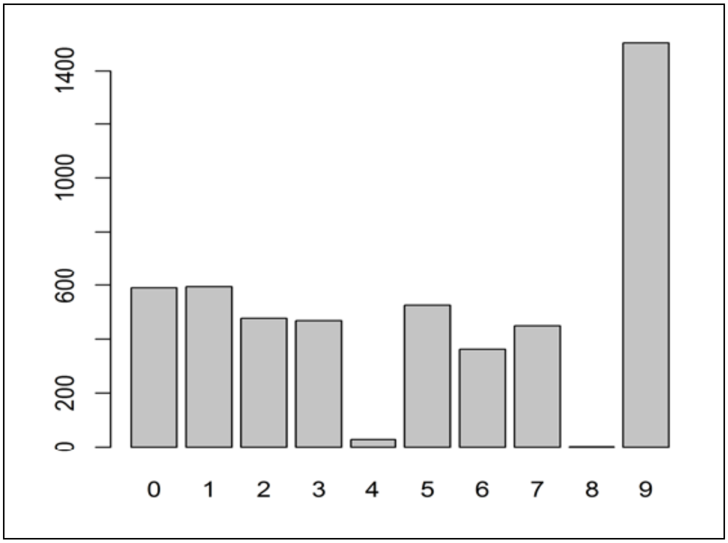


图 2-3

```
set.seed(1234)
digits.m2 <- train(digits.X, digits.y,
           method = "nnet",
           tuneGrid = expand.grid(
             .size = c(10),
             .decay = 0.1),
           trControl = trainControl(method = "none"),
            MaxNWts = 50000,
            maxit = 100)

digits.yhat2 <- predict(digits.m2)
barplot(table(digits.yhat2))
```

```
caret::confusionMatrix(xtabs(~digits.yhat2 + digits.y))

Confusion Matrix and Statistics

             digits.y
digits.yhat2   0    1    2    3    4    5    6    7    8    9
           0 395    0   14   23    0  120    6   12   15    5
           1   2  518   35   10    0    7    0   10    8    4
           2  23   23  323   15    8   37   30    1   15    2
           3   0    4   24  337    0   49    0   12   37    5
           4   3    0    0    0   10   14    2    0    0    0
           5  44    0   20   60    0  146   10    1  235    9
           6   1    1   25    2    0    3  327    0    3    0
           7   3    1    7    3    3   11    7  392    3   19
           8   0    0    0    0    0    0    1    0    0    0
           9  23   11   97   30  456   82  133   78  161  434

Overall Statistics

               Accuracy : 0.5764
                 95% CI : (0.5626, 0.5901)
    No Information Rate : 0.1116
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5293
Mcnemar's Test P-Value  : NA

Statistics by Class:

                  Class: 0 Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity         0.7996   0.9283   0.5927   0.7021  0.02096
Specificity         0.9567   0.9829   0.9654   0.9710  0.99580
Pos Pred Value      0.6695   0.8721   0.6771   0.7201  0.34483
Neg Pred Value      0.9776   0.9909   0.9509   0.9684  0.90606
Prevalence          0.0988   0.1116   0.1090   0.0960  0.09540
Detection Rate      0.0790   0.1036   0.0646   0.0674  0.00200
```

```
Detection Prevalence 0.1180   0.1188   0.0954   0.0936  0.00580
Balanced Accuracy    0.8782   0.9556   0.7790   0.8366  0.50838
                   Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity          0.3113   0.6337   0.7747   0.0000   0.9079
Specificity          0.9164   0.9922   0.9873   0.9998   0.7632
Pos Pred Value       0.2781   0.9033   0.8731   0.0000   0.2884
Neg Pred Value       0.9278   0.9592   0.9750   0.9046   0.9874
Prevalence           0.0938   0.1032   0.1012   0.0954   0.0956
Detection Rate       0.0292   0.0654   0.0784   0.0000   0.0868
Detection Prevalence 0.1050   0.0724   0.0898   0.0002   0.3010
Balanced Accuracy    0.6138   0.8130   0.8810   0.4999   0.8356
```

隐藏神经元的数量从 5 个增加到 10 个,样本内性能的总准确度从 44.3%提升到了 57.6%，但这与理想情形仍有相当大的距离（想象一下字符识别软件会弄混全部字符的 42.4%!)。这一次，我们再把隐藏神经元的数量增加到 40 个，完成这个模型的训练还要等更长的时间。

```
set.seed(1234)
digits.m3 <- train(digits.X, digits.y,
           method = "nnet",
           tuneGrid = expand.grid(
             .size = c(40),
             .decay = 0.1),
           trControl = trainControl(method = "none"),
           MaxNWts = 50000,
           maxit = 100)
digits.yhat3 <- predict(digits.m3)
barplot(table(digits.yhat3))
caret::confusionMatrix(xtabs(~digits.yhat3 + digits.y))

Confusion Matrix and Statistics

             digits.y
digits.yhat3    0    1    2    3    4    5    6    7    8    9
           0  461    0    7    3    0   20   16    2    3    7
```

```
1     0   521     3     4     0     2     2     6    10     2
2    17     3   469    30     2    13    16    10    39     2
3     1     5    11   352     2    43     2     9    48     5
4     1     0     6     1   394     7     0     4     3    36
5     3     4     2    23     1   334    12     1    51     6
6     6     1    19     3    15    10   455     1     3     1
7     0     2     8     7     5     5     2   411     6    35
8     2    20    10    46     4    28     9    10   297    23
9     3     2    10    11    54     7     2    52    17   361
```

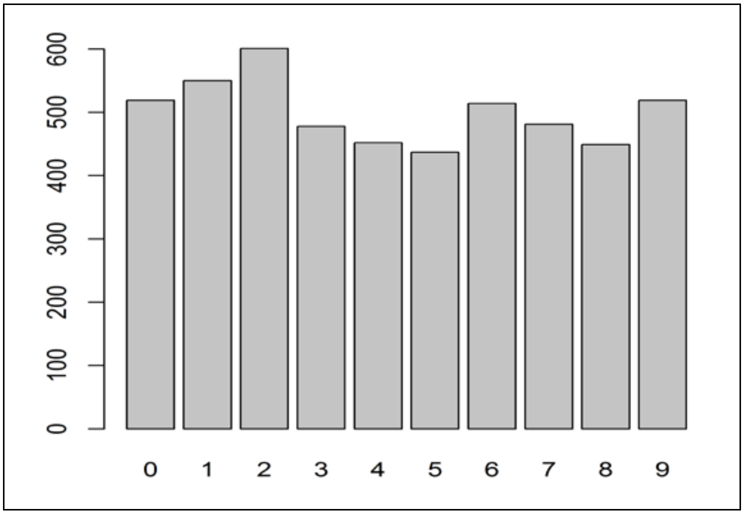


图 2-4

```
Overall Statistics

               Accuracy : 0.811
                 95% CI : (0.7999, 0.8218)
    No Information Rate : 0.1116
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7899
 Mcnemar's Test P-Value : NA

Statistics by Class:
```

```
                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.9332   0.9337   0.8606   0.7333   0.8260
Specificity            0.9871   0.9935   0.9704   0.9721   0.9872
Pos Pred Value         0.8882   0.9473   0.7804   0.7364   0.8717
Neg Pred Value         0.9926   0.9917   0.9827   0.9717   0.9818
Prevalence             0.0988   0.1116   0.1090   0.0960   0.0954
Detection Rate         0.0922   0.1042   0.0938   0.0704   0.0788
Detection Prevalence 0.1038   0.1100   0.1202   0.0956   0.0904
Balanced Accuracy      0.9602   0.9636   0.9155   0.8527   0.9066
                     Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity            0.7122   0.8818   0.8123   0.6226   0.7552
Specificity            0.9773   0.9868   0.9844   0.9664   0.9651
Pos Pred Value         0.7643   0.8852   0.8545   0.6615   0.6956
Neg Pred Value         0.9704   0.9864   0.9790   0.9604   0.9739
Prevalence             0.0938   0.1032   0.1012   0.0954   0.0956
Detection Rate         0.0668   0.0910   0.0822   0.0594   0.0722
Detection Prevalence 0.0874   0.1028   0.0962   0.0898   0.1038
Balanced Accuracy      0.8447   0.9343   0.8983   0.7945   0.8601
```

我们使用了 40 个隐藏神经元后，性能的提高引人注目，总准确度上升到 81.1%。模型性能对数字 3、5、8 和 9 还不是很优异，但对其他数字相当好。如果这是一项实际研究或商业问题，我们会继续尝试增加神经元的数量，调节衰变率，或者为了提升性能而修正特征，但现在，我们的话题将继续深入。

接下来，我们看看如何使用 RSNNS 包训练神经网络。对于可能使用斯图加特神经网络仿真器（Stuttgart Neural Network Simulator，SNNS）代码的诸多模型，这个包提供了相应的接口。但是，对基本的、单隐藏层的、前馈的神经网络，我们可以使用 mlp() 这个更为方便的封装函数，它的名称表示多层感知器（multi-layer perceptron）。相比于方便的 nnet 包，RSNNS 包的使用要通过 caret 包，难度相对较高，但优点在于更灵活并且可以训练更多的神经网络架构，包括循环神经网络以及种类更广泛的学习函数。

nnet 包和 RSNNS 包之间的区别在于结果的多分类（比如几个数字），RSNNS 要求一个哑编码的矩阵，因此每个可能的类表示成矩阵列中的 0/1 编码。使用 decodeClassLabels()函数可以很容易做到，接下来显示了一些输出。

```
head(decodeClassLabels(digits.y))

     0 1 2 3 4 5 6 7 8 9
[1,] 0 1 0 0 0 0 0 0 0 0
[2,] 1 0 0 0 0 0 0 0 0 0
[3,] 0 1 0 0 0 0 0 0 0 0
[4,] 0 0 0 0 1 0 0 0 0 0
[5,] 1 0 0 0 0 0 0 0 0 0
[6,] 1 0 0 0 0 0 0 0 0 0
```

因为使用 40 个隐藏神经元后已经获得了相当的成功，这里我们使用相同的数目。基于 Riedmiller, M.和 Braun, H.（1993）的经典工作，我们使用弹性传播作为学习函数，而非标准传播。注意，因为传递了一个矩阵的结果，尽管任何单一数字的预测概率不会超过 1，所有数字的预测概率和也许会超过 1 也许会小于 1（即在某些情况下，这些模型未必能预测，它们很可能代表任何一个数字）。和之前一样，我们可以得到样本内预测，但在这里我们要使用其他函数，`fitted.values()`。因为这一次还是返回一个矩阵，每列代表单个数字，我们使用 `encodeClassLabels()` 函数把结果转化成数字标签的单个向量从而画图，如图 2-5 所示，并评价模型性能。

```
set.seed(1234)
digits.m4 <- mlp(as.matrix(digits.X),
             decodeClassLabels(digits.y),
             size = 40,
             learnFunc = "Rprop",
             shufflePatterns = FALSE,
             maxit = 60)
digits.yhat4 <- fitted.values(digits.m4)
digits.yhat4 <- encodeClassLabels(digits.yhat4)
barplot(table(digits.yhat4))
```

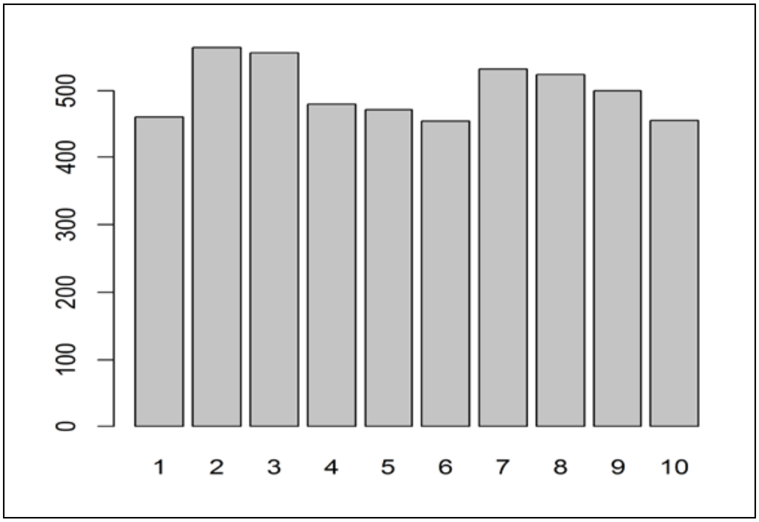


图 2-5

一旦我们预测了概率，使用 nnet 包和 caret 包评价模型性能就完全一样了。唯一的隐患是，当输出被编码返回单个向量时，默认数字标记为 1 到 k，其中 k 是类别个数。因为数字是从 0 到 9，为了使它们匹配原始数值向量，我们减去了 1。接下来我们可以看到，使用 RSNNS 包的学习算法并使用了相同数目的隐藏神经元，我们的计算结果的性能稍有提高。接下来，我们预测样本外数据。

```
caret::confusionMatrix(xtabs(~ I(digits.yhat4 - 1) + digits.y))

Confusion Matrix and Statistics

                    digits.y
I(digits.yhat4 - 1)    0    1    2    3    4    5    6    7    8    9
                  0  451    0    0    1    0    2    3    2    1    1
                  1    0  534    4    2    3    1    0    7   11    2
                  2    6    3  496   17    3    4    2    4   20    1
                  3    9    5   11  406    3   21    0    2   13   10
                  4    2    1    6    0  415    7    4    4    9   24
                  5   12    2    0   14    3  376    8    4   23   13
                  6    4    4    2    2    3   12  493    2    9    1
```

```
          7    3    0   10    7    4    1    1  460    1   37
          8    5    9   14   28   12   31    5    8  375   13
          9    2    0    2    3   31   14    0   13   15  376

Overall Statistics

               Accuracy : 0.8764
                 95% CI : (0.867, 0.8854)
    No Information Rate : 0.1116
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8626
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.9130   0.9570   0.9101   0.8458   0.8700
Specificity            0.9978   0.9932   0.9865   0.9836   0.9874
Pos Pred Value         0.9783   0.9468   0.8921   0.8458   0.8792
Neg Pred Value         0.9905   0.9946   0.9890   0.9836   0.9863
Prevalence             0.0988   0.1116   0.1090   0.0960   0.0954
Detection Rate         0.0902   0.1068   0.0992   0.0812   0.0830
Detection Prevalence 0.0922   0.1128   0.1112   0.0960   0.0944
Balanced Accuracy      0.9554   0.9751   0.9483   0.9147   0.9287
                      Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity            0.8017   0.9554   0.9091   0.7862   0.7866
Specificity            0.9826   0.9913   0.9858   0.9724   0.9823
Pos Pred Value         0.8264   0.9267   0.8779   0.7500   0.8246
Neg Pred Value         0.9795   0.9949   0.9897   0.9773   0.9776
Prevalence             0.0938   0.1032   0.1012   0.0954   0.0956
Detection Rate         0.0752   0.0986   0.0920   0.0750   0.0752
Detection Prevalence 0.0910   0.1064   0.1048   0.1000   0.0912
Balanced Accuracy      0.8921   0.9734   0.9474   0.8793   0.8845
```

2.1.2 从神经网络生成预测

到现在为止，我们仅仅在用于训练神经网络的相同数据上生成了样本内预测，而且为了获得分类接受了所有的缺省配置。然而，即使模型受到了训练，实际上也存在一些选择。任何给定的观测都有概率成为任何一个类别成员（例如，一个观测有 40%的概率成为“5”，20%的概率成为“6”，等等）。为了评价模型性能，对于如何从类别成员的概率映射到离散分类，我们需要做出一些选择。在本节中，我们将更加细致地探索一些这样的参数，同时在没有用于训练模型的数据上看看生成的预测。

只要完美关系不存在，最简单的方法就是基于高预测概率来对观测进行分类。另有一种方法，RSNNS 包称为赢者通吃（winner takes all，WTA）方法，这是指只要没有关系就选择概率最高的类，最高的概率高于用户定义的阈值（这个阈值可以是 0），而其他类的预测概率都低于最大值减去另一个用户定义的阈值。否则观测的分类就不明了。如果这两个阈值都是 0（缺省），这等于说最大值必然存在并且唯一。这种方法的优点是它提供了某种质量控制。在我们曾经探索的数字分类例子中，存在 10 种可能的分类。假设当中的 9 个有 0.099 的预测概率，其余的一类有 0.101 的预测概率。尽管有一类比其他各类在技术上更合适，这种差异太微小，而且可以推断模型无法区分出这个预测。最后一种分类方法称为“402040”，这是指如果一个值高于一个用户定义的阈值，而所有的其他值低于用户定义的另一个阈值。如果多个值都高于第一个阈值，或者任何值都不低于第二个阈值，我们就把观测定性为未知的。这样做的目的是再次给出了某种质量控制。看起来这可能并不必要，因为预测的不确定性会出现在模型性能中。但是，它有助于明确我们的模型对错是否确定。最后，在有些情况下，并非所有类的重要性都相同。例如，在一个医学背景下，我们收集了病人的多种生物指标和基因信息，用来分类确定他们是否健康，是否有患癌症的风险，是否有患心脏病的风险，即使有 40%的患癌概率也需要病人进一步做检查，即使健康的概率是 60%。这与我们之前见过的性能度量有关，在那些性能度量中，除了总准确度，我们能评估诸如灵敏度、特异度、正预测值和负预测值这些

方面的性能。这是总准确度比确保无遗漏次要的情况。

下面的代码展示了样本内数据的原始概率以及这些不同的选择对预测值的影响。

```
digits.yhat4.insample <- fitted.values(digits.m4)
head(round(digits.yhat4.insample, 2))

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 0.00 0.89 0.00 0.01 0.00 0.00 0.00 0.00 0.21     0
[2,] 0.99 0.00 0.00 0.02 0.00 0.00 0.00 0.01 0.00     0
[3,] 0.00 1.00 0.09 0.00 0.00 0.00 0.00 0.05 0.00     0
[4,] 0.00 0.00 0.00 0.00 0.22 0.00 0.02 0.05 0.00     0
[5,] 1.00 0.00 0.02 0.00 0.00 0.00 0.00 0.00 0.00     0
[6,] 0.99 0.00 0.00 0.00 0.00 0.06 0.00 0.00 0.00     0

table(encodeClassLabels(digits.yhat4.insample,
                        method = "WTA", l = 0, h = 0))

  1   2   3   4   5   6   7   8   9  10
461 564 556 480 472 455 532 524 500 456

table(encodeClassLabels(digits.yhat4.insample,
                        method = "WTA", l = 0, h = .5))

  0   1   2   3   4   5   6   7   8   9  10
569 448 544 497 400 429 366 499 463 379 406

table(encodeClassLabels(digits.yhat4.insample,
                        method = "WTA", l = .2, h = .5))

  0   1   2   3   4   5   6   7   8   9  10
658 443 542 490 393 408 358 493 460 364 391

table(encodeClassLabels(digits.yhat4.insample,
                        method = "402040", l = .4, h = .6))
```

```
  0   1   2   3   4   5   6   7   8   9  10
907 431 526 472 363 383 326 475 448 301 368
```

使用 predict()函数，我们可以很容易生成预测值。对此，我们将使用接下来的 5 000 个观测。注意，即使在一个新桌面上生成这些预测也需要花费几分钟的时间。

```
i2 <- 5001:10000
digits.yhat4.pred <- predict(digits.m4,
                             as.matrix(digits.train[i2, -1]))

table(encodeClassLabels(digits.yhat4.pred,
                        method = "WTA", l = 0, h = 0))

  1   2   3   4   5   6   7   8   9  10
449 570 531 518 476 442 522 533 468 491
```

生成了样本外数据（就是说，那些没用于预测模型的数据）的预测之后，现在我们可以转向检查有关数据过拟合的问题以及对评价模型性能的影响。

2.2 数据过拟合的问题——结果的解释

机器学习的一个常见议题是数据过拟合的问题。通常来说，过拟合指这样一种现象，训练模型的数据的模型性能优于未使用训练模型的数据［保留数据（holdout data）、未来真正使用的数据等］。过拟合发生在模型正好拟合了训练数据的噪声部分的时候。因为考虑了噪声，它似乎更准确，但一个数据集和下一个数据集的噪声不同，这种准确度不能运用于除了训练数据之外的任何数据——它没有一般化。

过拟合可以发生在任何时间，但随着参数对信息的比例上升往往会恶化。通常

来说，这个比例可以认为是参数对观测的比例，但并不总是如此（例如，假定结果是在 500 万人中发生一次的一个罕见事件，一个 1 500 万的样本规模也仅仅有 3 个人会体验到这种事件，并且根本不支持任何的复杂模型——即使样本规模很大但信息很低）。我们考虑一个简单但极端的情况，想象对两个数据点拟合一条直线。拟合会很完美，而且在这两个训练数据中线性回归模型看起来充分考虑了数据的所有变化。但是，如果我们把这条线运用到另 1 000 个实例上，我们完全不会期待它拟合得有多好。

在 2.1 节，我们对训练的 RSNNS 模型生成了样本外预测。我们知道，样本内的准确度是 87.6%。这个估计有多好？我们使用现在已经熟悉的代码检查样本外预测的精度，可以检查这个模型一般化的程度。接下来我们可以看到，它的程度相当好，但保留数据的准确度减少到了 83.6%。看起来损失接近 4%，换句话说，使用训练数据来评价模型性能导致了过度乐观的准确度估计，过度估计是 4%。

```
caret::confusionMatrix(xtabs(~digits.train[i2, 1] +
  I(encodeClassLabels(digits.yhat4.pred) - 1)))
Confusion Matrix and Statistics

                   I(encodeClassLabels(digits.yhat4.pred) - 1)
digits.train[i2, 1]   0   1   2   3   4   5   6   7   8   9
                  0 429   0  13  16   4   9   8   4   9   5
                  1   0 515   9   3   0   2   2   2   4   0
                  2   4   7 427  17   2   3  12  10  12   6
                  3   0   2  20 416   2  28   5  11  40   5
                  4   0   6   6   8 392   7  13   2  19  37
                  5   8   2   4  24  15 335  11   7  21  10
                  6   2   1   8   1   1   9 460   0   3   2
                  7   1  14  22   9   8   2   2 459   3  13
                  8   4  23  19  11  16  27   8   5 348  12
                  9   1   0   3  13  36  20   1  33   9 401

Overall Statistics
```

```
                Accuracy : 0.836
                  95% CI : (0.826, 0.847)
     No Information Rate : 0.114
     P-Value [Acc > NIR] : <2e-16

                   Kappa : 0.818
  Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.9555    0.904   0.8041   0.8031   0.8235
Specificity            0.9851    0.995   0.9837   0.9748   0.9783
Pos Pred Value         0.8632    0.959   0.8540   0.7864   0.8000
Neg Pred Value         0.9956    0.988   0.9769   0.9772   0.9814
Prevalence             0.0898    0.114   0.1062   0.1036   0.0952
Detection Rate         0.0858    0.103   0.0854   0.0832   0.0784
Detection Prevalence 0.0994    0.107   0.1000   0.1058   0.0980
Balanced Accuracy      0.9703    0.949   0.8939   0.8889   0.9009
                     Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity            0.7579   0.8812   0.8612   0.7436   0.8167
Specificity            0.9776   0.9940   0.9834   0.9724   0.9743
Pos Pred Value         0.7666   0.9446   0.8612   0.7357   0.7756
Neg Pred Value         0.9766   0.9863   0.9834   0.9735   0.9799
Prevalence             0.0884   0.1044   0.1066   0.0936   0.0982
Detection Rate         0.0670   0.0920   0.0918   0.0696   0.0802
Detection Prevalence 0.0874   0.0974   0.1066   0.0946   0.1034
Balanced Accuracy      0.8678   0.9376   0.9223   0.8580   0.8955
```

因为我们以前拟合了几个复杂程度不同的模型，通过对比它们的样本内和样本外度量，可以检查过拟合的程度或者过度乐观的准确度。代码没有显示，因为只是我们已经运行过的代码的重复，但我们可以从本书提供的代码包中得到它。结果显示如图 2-6 所示。

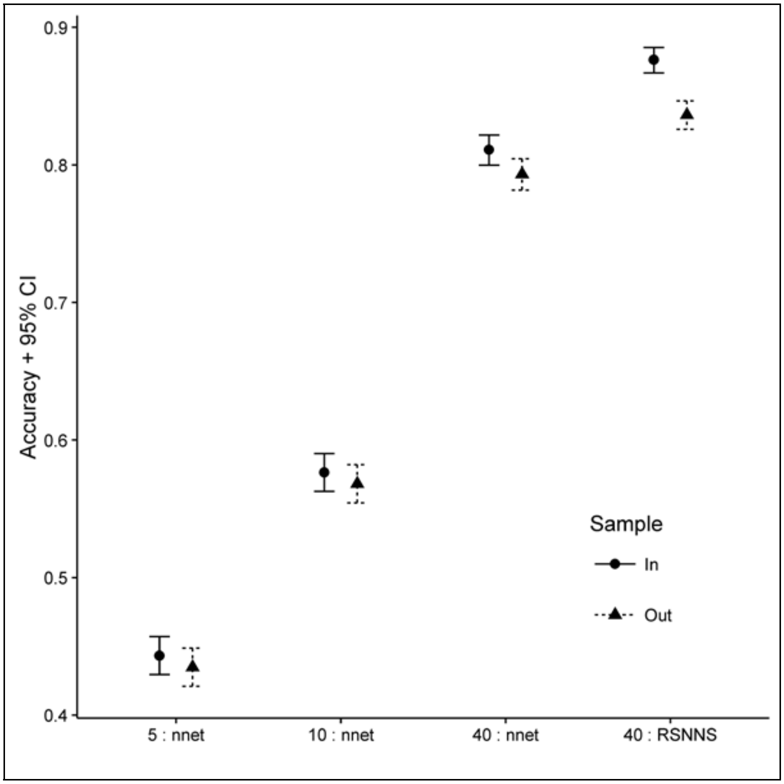


图 2-6

2.3 用例——建立并运用神经网络

在本章的最后，我们来讨论神经网络的一个更实际的用例。我们用 Anguita, D., Ghio, A., Oneto, L., Parra, X.,和 Reyes-Ortiz, J. L. （2013）提供的公开数据，使用智能手机追踪体育活动。数据可以从以下地址下载：`http://archive.ics.uci.edu/ml/ datasets/Human+Activity+ Recognition+Using+ Smartphones`。智能手机有加速计和陀螺仪，可以从时间和频率的 561 个特征中得到。

用户在行走、上楼、下楼、站立、坐下和躺倒时，都把智能手机带在身上。尽

管这个数据来自于手机，类似的测量也可以从用于追踪活动设计的设备采集，比如各种健康追踪手表或腕带。所以如果我们希望销售设备并且用它们自动追踪佩戴者从事了多少不同运动，这个数据会很有用。

这个数据已经标准化，范围从−1 到 1。但是，通常我们可能会运行一些标准化。数据下载之后，我们可以解压文件并且把它们存放到工作目录中，或者在下列代码中修改路径指向正确的位置。我们可以读入训练数据和测试数据以及标签，同时整体观察结果的分布，如图 2-7 所示。

```
use.train.x <- read.table("UCI HAR Dataset/train/X_train.txt")
use.train.y <- read.table("UCI HAR Dataset/train/y_train.txt")
[[1]]

use.test.x <- read.table("UCI HAR Dataset/test/X_test.txt")
use.test.y <- read.table("UCI HAR Dataset/test/y_test.txt")[[1]]

use.labels <- read.table("UCI HAR Dataset/activity_labels.txt")

barplot(table(use.train.y))
```

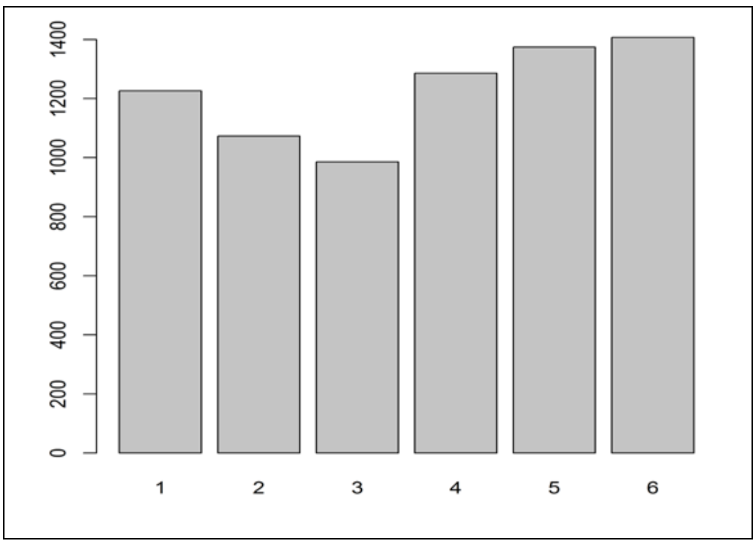


图 2-7

为了展示如何使用不同的方法来尽量得到最好的模型，我们将评价各种调节参数。因为模型需要一些时间来训练并且目前仅显示使用了单核，我们可以使用并行过程，同时使用不同的调节参数来评价模型。首先，我们需要加入一些额外的包到 checkpoint.R 文件中并重新运行。

```
## Chapter 2 ##
library(parallel)
library(foreach)
library(doSNOW)
```

现在我们能挑选调节参数，再建立一个本地集群，作为 R 包 foreach 的后台，用来使用并行 `for` 循环。注意，如果我们使用的机器少于 5 核，需要把 `makeCluster(5)`改为更小的数。

```
## choose tuning parameters
tuning <- list(
  size = c(40, 20, 20, 50, 50),
  maxit = c(60, 100, 100, 100, 100),
  shuffle = c(FALSE, FALSE, TRUE, FALSE, FALSE),
  params = list(FALSE, FALSE, FALSE, FALSE, c(0.1, 20, 3)))

## setup cluster using 5 cores
## load packages, export required data and variables
## and register as a backend for use with the foreach package
cl <- makeCluster(5)
clusterEvalQ(cl, {
  library(RSNNS)
})
clusterExport(cl,
  c("tuning", "use.train.x", "use.train.y",
    "use.test.x", "use.test.y")
  )
registerDoSNOW(cl)
```

现在我们已经准备好训练所有的模型。下面的代码提供了一个并行的 for 循环，使用类似于我们已经见过的代码，但这次是基于我们之前存储在列表中的调节参数来设置一些参数。

```
use.models <- foreach(i = 1:5, .combine = 'c') %dopar% {
  if (tuning$params[[i]][1]) {
    set.seed(1234)
    list(Model = mlp(
      as.matrix(use.train.x),
      decodeClassLabels(use.train.y),
      size = tuning$size[[i]],
      learnFunc = "Rprop",
      shufflePatterns = tuning$shuffle[[i]],
      learnFuncParams = tuning$params[[i]],
      maxit = tuning$maxit[[i]]
      ))
    } else {
      set.seed(1234)
      list(Model = mlp(
        as.matrix(use.train.x),
        decodeClassLabels(use.train.y),
        size = tuning$size[[i]],
        learnFunc = "Rprop",
        shufflePatterns = tuning$shuffle[[i]],
        maxit = tuning$maxit[[i]]
      ))
    }
}
```

因为生成样本外预测也会需要一些时间，我们也用并行来处理。首先，我们需要把模型结果输出到集群上的每个 worker 节点，然后我们可以计算这个预测。

```
clusterExport(cl, "use.models")
use.yhat <- foreach(i = 1:5, .combine = 'c') %dopar% {
  list(list(
```

```
    Insample = encodeClassLabels(fitted.values(use.models[[i]])),
    Outsample = encodeClassLabels(predict(use.models[[i]],
                                          newdata = as.matrix
(use.test.x)))
    ))
}
```

最后，我们可以把实际的和拟合或预测的值并入一个数据集，在每个值上计算性能度量，再把所有的结果一起存放进行检查和比较。我们可以重复与如下代码几乎完全相同的代码来生成样本外的性能度量。这些代码在本书中不出现，但我们可以从本书提供的代码包中得到。这里需要一些额外的数据管理，因为模型有时无法预测出每一个可能的对应水平，但可以做出非对称的频率交叉表，除非我们把变量转化为因子并指定水平。我们也会放弃 0 值，这个值表示模型不确定如何对一个观测归类。

```
use.insample <- cbind(Y = use.train.y,
  do.call(cbind.data.frame, lapply(use.yhat, '[[', "Insample")))
colnames(use.insample) <- c("Y", paste0("Yhat", 1:5))

performance.insample <- do.call(rbind, lapply(1:5, function(i) {
  f <- substitute(~ Y + x, list(x = as.name(paste0("Yhat", i))))
  use.dat <- use.insample[use.insample[,paste0("Yhat", i)]!= 0, ]
  use.dat$Y <- factor(use.dat$Y, levels = 1:6)
  use.dat[, paste0("Yhat", i)]<- factor(use.dat[, paste0("Yhat",
i)],
levels = 1:6)
  res <- caret::confusionMatrix(xtabs(f, data = use.dat))

  cbind(Size = tuning$size[[i]],
       Maxit = tuning$maxit[[i]],
       Shuffle = tuning$shuffle[[i]],
       as.data.frame(t(res$overall[c("AccuracyNull", "Accuracy",
"AccuracyLower", "AccuracyUpper")])))
}))
```

如果打印出样本内和样本外的性能，我们可以看到每个模型和调节参数变化的影响。输出显示在下列代码中。我们放弃了第 4 列（零准确度，null accuracy），因为它对于这种比较相对次要。注意，样本外性能的代码没有在本书中显示，但作为练习题留给大家（很容易根据样本内性能的代码来改编），在代码包中有提供。

```
performance.insample[,-4]

  Size Maxit Shuffle Accuracy AccuracyLower AccuracyUpper
1   40    60   FALSE     0.99          0.98          0.99
2   20   100   FALSE     0.99          0.99          0.99
3   20   100    TRUE     0.99          0.99          0.99
4   50   100   FALSE     0.99          0.99          1.00
5   50   100   FALSE     1.00          1.00          1.00

performance.outsample[,-4]
  Size Maxit Shuffle Accuracy AccuracyLower AccuracyUpper
1   40    60   FALSE     0.93          0.92          0.94
2   20   100   FALSE     0.92          0.91          0.93
3   20   100    TRUE     0.92          0.91          0.93
4   50   100   FALSE     0.91          0.90          0.92
5   50   100   FALSE     0.92          0.91          0.93
```

首先，这些结果显示，我们能够根据智能手机的数据非常准确地对人们从事的活动分类。样本内数据似乎告诉我们越复杂的模型越好。但是，我们通过检查样本外的性能度量，发现事实上情况正好相反！因此，样本内性能度量不仅是模型的真实样本外性能的有偏估计，它们甚至无法对模型性能排序提供最佳方式，从而用来选择性能最好的模型。我们将在第 3 章中讨论解决过拟合问题的方法，同时准备讨论有多个隐藏层的深度神经网络。

尽管样本外的性能稍差一些，模型依然运行良好，比仅仅通过可能性分类更好。而且，对于我们的用例，我们可以挑选出最佳模型（第 1 号），而且很有信心使用这个模型会提供用户活动的很好分类。

2.4 小结

本章介绍了如何开始建立并训练神经网络模型，用来对包括图像识别和体育活动数据在内的数据进行分类。机器学习的一个陷阱是，越复杂的数据越有可能过拟合训练数据，因此，对相同数据训练模型的性能评价会导致有偏的、过度乐观的模型性能的估计。事实上，这甚至会影响到哪个模型被选为最佳。过拟合对于深度神经网络来说也是一个问题，在第 3 章中，我们将讨论几种防止过拟合的方法，称为正则化，从而获得模型性能的更准确估计。

第 3 章 防止过拟合

在第 2 章中，我们学习了如何训练一个基本的神经网络。对于用于模型训练的留存数据进行验证后，我们还看到，进一步的训练迭代或者更大的神经网络产生的收益都在递减。这里强调的是，尽管一个更复杂的模型几乎总是会把训练它的数据拟合得更好，但它未必能把新数据预测得更好。本章介绍为了提升泛化能力而用于防止数据过拟合的不同的方法，称为无监督数据上的正则化（regularization on unsupervised data）。更具体地说，与通常地按照减少训练（training）误差的方式来优化参数训练模型不同，正则化关注于减少测试（testing）或验证（validation）误差，这样模型在新数据上的性能会和在训练数据上的一样好。

本章的开始提供了各种正则化策略的一个概念性的综述，以一个使用正则化来提升样本外性能的用例结束。它包含了下面的主题。

- L1 罚函数
- L2 罚函数
- 集成方法与模型平均
- 用例——使用丢弃提升样本外的模型性能

3.1　L1 罚函数

L1 罚函数，也称为最小绝对值收缩和选择算子（Least Absolute Shrinkage and Selection Operator，lasso）（Hastie, T., Tibshirani, R.和 Friedman, J.（2009）），它的基本思想是一种用来把权重向零的方向缩减的惩罚。惩罚项使用的是权重绝对值的和，所以无论对于小的还是大的权重，惩罚的程度不会更小或者更大，结果是小的权重会缩减到零，作为一种方便的效果，除了防止过拟合之外，它还可以作为一种变量选择的方法。惩罚的力度是由一个超参数λ所控制的，它乘以权重绝对值的和，可以被预先设定，或者就像其他超参数那样，使用交叉验证或者一些类似的方法来优化。

就数学上来讲，从普通最小二乘（Ordinary Least Squares，OLS）回归开始介绍要更容易一些。在回归当中，我们使用最小二乘误差准则来估计一组系数或模型权重，其中权重/系数向量 B 通过最小化（$Y-XB$）^{T}（$Y-XB$）估计出来，Y 是结果或因变量，是一个列的设计矩阵，列对应预测变量，一个常数列对应截距（有时也叫作偏移）。观测的结果和预测值（设计矩阵乘以系数向量的积）之间相差一个误差或者残差向量。在这个框架中，L1 罚函数可以认为是一个有约束的估计变量，其中权重向量的估计满足绝对值权重的和小于等于某个（用户指定的）阈值λ的约束。

通常，截距或偏移项会从这个约束中排除（例如，通过预先中心化所有数据并且去掉截距，或者通过有选择地运用约束来实现这一点）。另一方面我们可以将 L1 罚函数看作函数最小化的一种修正，从（$Y-XB$）^{T}（$Y-XB$）到（$Y-XB$）^{T}（$Y-XB$）$+\lambda\|B\|_1$，其中$\|B\|_1$ 代表权重绝对值的和。如果$\lambda=0$，那么 L1 罚函数缩小到规则的 OLS 估计子。用户可以选择λ，或者更通俗地将它作为一个超参数，通过评价λ可能取值的一个范围（比如，通过交叉验证）进行优化。尽管超出了本书范围，L1 罚函数还可以通过贝叶斯观点来看，最终的后验估计是来自数据和先验估计的函数，可以通过设定一个有不同程度确定性的先验来实现产生于惩罚项的缩减。从技术上

来说，参数可以向任意的值收缩，但是它们几乎总是向零收缩。

即使对于 L1 罚函数为什么以及如何起作用背后的理论还不是很清楚，这里一些实践的含义是直接的。首先，很明显惩罚的影响依赖于权重的大小，而权重的大小依赖于数据的规模。因此，我们通常先把数据标准化为带有单位方差（或者起码是每个变量的方差相等）的形式。L1 罚函数有一种趋势，把小的权重向零的方向缩减（要解释为什么会发生这种情况，参见 Hastie, T., Tibshirani, R.和 Friedman, J.，(2009)）。如果我们只考虑那些 L1 罚函数留下的非零权重的变量，它本质上具有特征选择的功能，这是经常使用 L1 罚函数的另一个名称——最小绝对值收缩和选择算子，或者 lasso 的主要动机。即使在严格的特征选择使用之外，L1 罚函数把小的系数缩减到零的趋势，仍能方便地用来简化模型结果的解释。

把 L1 罚函数作为约束优化时，我们更容易看出它如何有效地限制了模型的复杂性。即使包括了许多预测变量，权重绝对值的和也不能超过定义的阈值。这样做的一个结果是，使用 L1 罚函数，只要有足够强的惩罚项，真的有可能包括比样例或观测还要多的预测变量。（根据权重个数）看起来过参数化的模型，通过这个约束变成唯一的估计。

有了 L1 罚函数的这些基础，现在我们主要考虑 L1 罚函数能怎样运用到神经网络上，它是我们在本书中关心的主要用例。我们用 X 表示输入，Y 表示输出或者因变量，B 是参数，F 是为了求出 B 而要优化的目标函数。特别地有：$F(B;X,Y)$。在神经网络中，参数可以是偏差或者偏移（本质上是来自回归的截距）以及权重。L1 罚函数把目标函数修正为其中 w 仅代表权重（就是说，偏移通常是被忽略的）。考虑梯度，我们可以将这个增加的惩罚项表示为 $F(B;X,Y)+\lambda\|w\|_{1}$。需要强调的是，不管权重的数量级如何，罚参数是一个常数。与我们接下来将要讨论的 L2 罚函数相比，这是一个重要的区别。进一步说，这是 L1 罚函数往往会导致稀疏结果（即更多的零权重）的方法的一部分。因为小的和更大的权重会导致相等的惩罚，所以梯度每次更新，权重会向零的方向移动。

我们已经讨论了，λ作为一个常数控制了惩罚或者正则化的程度。然而，设置不同的λ值是可能的。虽然这通常不在单层神经网络中实现（典型的是寻求有差别

的正则化的具体权重），但在深度学习网络中会变得更有用，其中正则化的变化程度可以运用到不同的层。考虑这种有差别的正则化的一种原因是，有时候我们想要允许更多的参数个数（在一个特定的层中包括更多的神经元）但之后通过更强的正则化在某种程度上抵消了这一点。尽管如此，既然这些参数通常是通过交叉验证或其他的经验技术优化的，为了允许它们对深度学习网络的每一层都变化，要有相当大的计算上的需求，因为可能的值个数是指数增长的。所以最常见的是在整个模型上使用单个的值。在 R 中探索 L1 罚函数的实践之后，我们会转而考虑另一种常见的正则化形式——L2 罚函数。

我们可以通过一个模拟的线性回归问题来看 L1 罚函数是如何工作的。首先，和以前一样，我们把 R 包 glmnet 添加到 checkpoint.R 文件中，用来载入相关的库并使用一个可重复的版本。

```
library(glmnet)
```

接下来我们可以模拟数据，使用一组有针对性的反常相关的预测变量的集合。

```
set.seed(1234)

X <- mvrnorm(n = 200, mu = c(0, 0, 0, 0, 0),
  Sigma = matrix(c(
    1, .9999, .99, .99, .10,
    .9999, 1, .99, .99, .10,
    .99, .99, 1, .99, .10,
    .99, .99, .99, 1, .10,
    .10, .10, .10, .10, 1
  ), ncol = 5))

y <- rnorm(200, 3 + X %*% matrix(c(1, 1, 1, 1, 0)), .5)
```

接下来，我们能对前 100 个样例拟合线性回归模型并且使用 lasso。为了使用 lasso，我们采用来自 glmnet 包的 `glmnet()` 函数。这个函数实际上能拟合 L1 罚函数或（后续我们会在 3.2 节讨论）L2 罚函数，取哪一个函数是由参数 `alpha` 来决

定的。当 alpha=1 时，它是 L1 罚函数（也就是 lasso）；当 alpha=2 时，它是 L2 罚函数（也就是岭回归）。而且，因为并不知道应该选取的 lambda 的值，我们能评价一系列的选择并且使用交叉验证自动调出这个超参数，这可以用 cv.glmnet() 函数来实现。

```
m.ols <- lm(y[1:100] ~ X[1:100, ])

m.lasso.cv <- cv.glmnet(X[1:100, ], y[1:100], alpha = 1)
```

我们能画出 lasso 对象，看看各个 lambda 值的均方误差。

```
plot(m.lasso.cv)
```

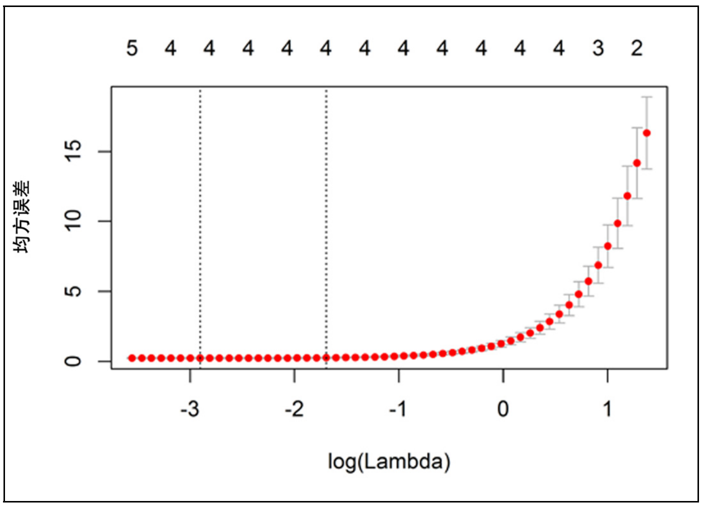


图 3-1

从图 3-1 我们能看到，当惩罚变得太大时，交叉验证模型的误差增加。事实上，lasso 看起来在非常低的 lambda 值上表现得很好，或许表明了 lasso 对于提高样本外的性能/泛化能力并不是很有帮助。为了完成这个例子，我们将会继续，但在实际的使用中这可能给了我们一个暂停的机会来思考 lasso 是否真的有帮助。

最后，我们用 OLS 系数和这些来自 lasso 的系数作比较。

```
cbind(
  OLS = coef(m.ols),
  Lasso = coef(m.lasso.cv)[,1])

                  OLS Lasso
(Intercept)  2.958  2.99
X[1:100, ]1 -0.082  1.41
X[1:100, ]2  2.239  0.71
X[1:100, ]3  0.602  0.51
X[1:100, ]4  1.235  1.17
X[1:100, ]5 -0.041  0.00
```

我们注意到 OLS 系数更杂乱，而在 lasso 当中，预测变量 5 被惩罚为 0。我们回忆模拟的数据，真实的系数是 3、1、1、1、1、0。OLS 估计对于第一个预测变量的值是太低了，对于第二个预测变量的值又太高了。反之，lasso 有更准确的值。

3.2 L2 罚函数

L2 罚函数，也叫作岭回归（ridge regression），除了惩罚是基于权重平方而不是基于权重绝对值的和之外，在许多方面是和 L1 罚函数很相似的。这样它就具有了提供不同惩罚的效果，更大的（正或者负）权重导致了更大的惩罚。在神经网络的背景下，这有时被称为权重衰减。如果我们检查正则目标函数的梯度，会发现存在一个惩罚，这样在每次更新中，对于权重有一个增加的惩罚。至于 L1 罚函数，尽管能够包含偏差或者偏移，通常也是被排除在外的。

从线性回归问题的观点，L2 罚函数是对目标函数最小化的修正，由 $(Y-XB)^{\mathrm{T}}(Y-XB)$修正为$(Y-XB)^{\mathrm{T}}(Y-XB)+0.5\lambda B^{\mathrm{T}}B$。和 L1 罚函数一样，L2 罚函数允许解决不同的不确定的问题，特别是当预测变量的协方差矩阵奇异的时候。这是因为，L2 罚函数的效果在本质上增加了每个变量的方差。在 OLS 中，B 的正规方程的矩阵形式是 $inv(X^{\mathrm{T}}X)X^{\mathrm{T}}y$，但求解之前所显示的正则的 OLS 目标函数，得到

$inv(X^TX+\lambda I)X^Ty$，其中是单位矩阵。

因为 X^TX 是设计矩阵的方差-协方差矩阵，增加λI 会有增加对角元素的效果，但非对角元素不变。这就是说，方差增加而协方差不变，导致相关系数（标准化的协方差）缩减到零。足够强的惩罚可以导致不同的奇异协方差矩阵成为唯一的估计，而且在有强相关预测变量的时候，这也有助于稳定估计。

3.2.1　L2 罚函数实战

来看 L2 罚函数是如何工作的，我们可以采用用在 L1 罚函数中的同一个模拟的线性回归问题。为了拟合岭回归模型，我们使用来自 glmnet 包的 `glmnet()`函数。和之前所提到的一样，这个函数实际上可以拟合 L1 罚函数或 L2 罚函数，取哪一个是由参数 `alpha` 决定的。当 `alpha=1` 时，它拟合 L1 罚函数（也就是 lasso）；当 `alpha=0` 时，它拟合岭回归。这一次我们选择 `alpha=0`。再一次，我们评价一系列的选择并且使用交叉验证自动地调出这个超参数，这通过 `cv.glmnet()`函数来实现。

```
m.ridge.cv <- cv.glmnet(X[1:100, ], y[1:100], alpha = 0)
```

我们画出岭回归对象，看看各个 lambda 值的均方误差。

```
plot(m.ridge.cv)
```

尽管这个形状和 lasso 不同，如图 3-2 中所示误差对更高的 lambda 值表现为渐进的，依然很清楚的是，当惩罚太高的时候，交叉验证模型误差增加。和 lasso 一样，岭回归往往在非常小的 lambda 值上表现很好，这可能表明 lasso 对于提高样本外性能/泛化能力并不是很有帮助。

最后，我们用 OLS 系数与这些来自 lasso 和岭回归的系数作比较。

```
cbind(
  OLS = coef(m.ols),
```

```
  Lasso = coef(m.lasso.cv)[,1],
  Ridge = coef(m.ridge.cv)[,1])
```

```
                    OLS   Lasso  Ridge
(Intercept)       2.958   2.99  3.002
X[1:100, ]1      -0.082   1.41  0.958
X[1:100, ]2       2.239   0.71  0.964
X[1:100, ]3       0.602   0.51  0.924
X[1:100, ]4       1.235   1.17  0.949
X[1:100, ]5      -0.041   0.00  0.011
```

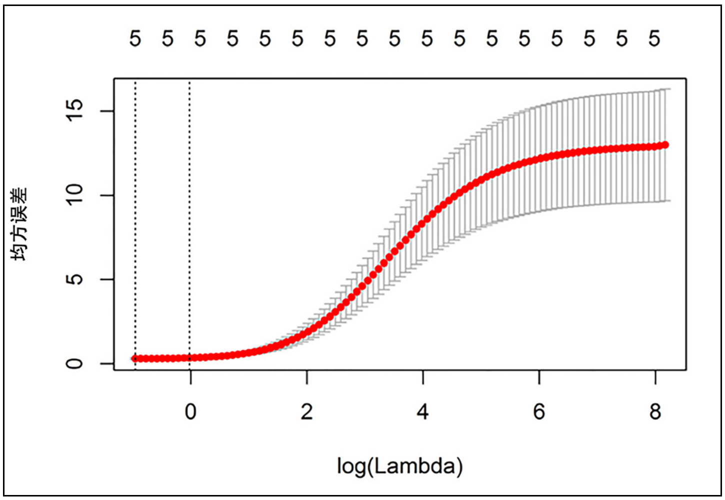


图 3-2

尽管岭回归没有把第五个预测子的系数缩减到精确的零，还是要比 OLS 的系数小，而且其余的参数都有轻微缩减，但是和它们的真实值 3、1、1、1、1、0 非常接近。

3.2.2 权重衰减（神经网络中的 L2 罚函数）

在了解权重衰减之前，我们实际上已经在第 2 章中看到了实战中的正则化。我

们使用 caret 包和 nnet 包训练的神经网络使用了 0.01 的权重衰减。为了探索权重衰减的使用，我们可以通过变化它的值以及使用交叉验证来调整取值。首先，我们和以前一样载入数据。然后，我们使用一个本地的集群，以并行的方式运行交叉验证。注意，和以前一样不是直接载入包，我们需要用 `source()` 调用 `checkpoint.R` 文件，这样才能保证集群中的每一个 worker 节点使用相同的 R 包版本。

```
## same data as from previous chapter
digits.train <- read.csv("train.csv")
## convert to factor
digits.train$label <- factor(digits.train$label, levels = 0:9)

i <- 1:5000
digits.X <- digits.train[i, -1]
digits.y <- digits.train[i, 1]

## try various weight decays and number of iterations
## register backend so that different decays can be
## estimated in parallel
cl <- makeCluster(4)
clusterEvalQ(cl, {
  source("checkpoint.R")
})
registerDoSNOW(cl)
```

接下来，我们在数字分类问题上建立一个神经网络，权重衰减在 0（没有惩罚）和 0.10 之间变化。我们分别循环迭代次数为 100 和 150 的两个集合。注意这个代码是计算密集型的而且依赖于硬件，可能要花一些时间来运行。

```
set.seed(1234)
digits.decay.m1 <- lapply(c(100, 150), function(its) {
  train(digits.X, digits.y,
            method = "nnet",
            tuneGrid = expand.grid(
               .size = c(10),
```

```
                 .decay = c(0, .1)),
              trControl = trainControl(method = "cv", number = 5,
repeats =1),
              MaxNWts = 10000,
              maxit = its)
   })
```

检查这个结果，我们可以看到，当迭代限制在 100 次的时候，基于交叉验证的结果，非正则化的模型（准确度=0.63）比正则化的模型（准确度=0.6）表现要好（尽管在这个数据上，两者做的都不好）。

```
digits.decay.m1[[1]]
Neural Network

5000 samples
 784 predictor
  10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 4000, 3999, 4000, 4001, 4000
Resampling results across tuning parameters:

    decay  Accuracy  Kappa   Accuracy SD  Kappa SD
    0.0    0.63      0.59    0.052        0.058
    0.1    0.60      0.56    0.061        0.068

Tuning parameter 'size' was held constant at a value of 10
Accuracy was used to select the optimal model using the
 largest value.
The final values used for the model were size = 10 and decay = 0.
```

接下来，我们可以检查 150 次迭代的模型，看看是正则化的模型还是非正则化的模型表现更好。

```
digits.decay.m1[[2]]
Neural Network

5000 samples
 784 predictor
  10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 4002, 4000, 4000, 3999, 3999
Resampling results across tuning parameters:

    decay  Accuracy  Kappa   Accuracy SD  Kappa SD
    0.0    0.65      0.61    0.049        0.055
    0.1    0.66      0.62    0.071        0.078

Tuning parameter 'size' was held constant at a value of 10
Accuracy was used to select the optimal model using the
 largest value.
The final values used for the model were size = 10 and decay = 0.1.
```

总的来说，不管是不是正则化的，更多迭代的模型比更少迭代的模型表现要好。然而，比较 150 次迭代的两个模型，正则化的模型（准确度=0.66）比非正则化的模型更好，尽管这里的差距相对小一些。

这些结果强调的重点是，正则化通常对更复杂的、有更大的灵活性拟合（以及过拟合）数据的模型最有用，同时（在那些对于数据合适的或者过于简单的模型）正则化实际上可能降低了性能。接下来，我们将讨论集成以及模型平均技术，这是我们将在本书中强调的最后一种正则化的形式。

3.3 集成和模型平均

另一种正则化的方法包括创建模型的集成并且把它们组合起来，例如，利用模型平均或者其他把个别模型的结果组合起来的算法。如同之前许多正则化的方法，模型平均是一种相当简单的概念。如果我们有不同的模型，每一个生成一组预测，每个模型也许会在预测中造成误差。一个模型可能把某个值预测得太高，另一个可能会把它预测得太低，这样平均起来，一些误差相互抵消，产生比通过其他方法更为准确的预测。

为了更好地理解模型平均，我们来假设两个不同但是极端的例子。在第一种情况中，我们假设被平均的模型是相同的，或者至少生成相同的预测（就是说，是完美相关的）。如果是那样，平均不会产生什么好处，但也没什么坏处。在第二种情况中，每个被平均的模型相互独立且同样好地被执行，而且它们的预测是不相关的（或者有非常低的相关性）。那么平均会精确得多得多，因为它获得了每一个模型的力量。下面的代码给出了一个使用模拟数据的例子。在这个小例子中，我们只有三个模型，但是它们说明了重点。

```
## simulated data
set.seed(1234)
d <- data.frame(
  x = rnorm(400))
d$y <- with(d, rnorm(400, 2 + ifelse(x < 0, x + x^2, x + x^2.5),
1))
d.train <- d[1:200, ]
d.test <- d[201:400, ]

## three different models
m1 <- lm(y ~ x, data = d.train)
m2 <- lm(y ~ I(x^2), data = d.train)
m3 <- lm(y ~ pmax(x, 0) + pmin(x, 0), data = d.train)
```

```
## In sample R2
cbind(
  M1 = summary(m1)$r.squared,
  M2 = summary(m2)$r.squared,
  M3 = summary(m3)$r.squared)

       M1   M2   M3
[1,] 0.33 0.60 0.76
```

我们能看到每个模型的预测值，至少在训练数据中变化是相当大的。在评价训练数据中，拟合值之间的相关性也能有助于识别在模型预测之间有着多大的重叠。

```
## correlations in the training data
cor(cbind(
  M1 = fitted(m1),
  M2 = fitted(m2),
  M3 = fitted(m3)))

     M1   M2   M3
M1 1.00 0.11 0.65
M2 0.11 1.00 0.78
M3 0.65 0.78 1.00
```

接下来，我们对测试数据生成预测值、预测值的平均值，而且再次求预测以及测试数据中真实情况的相关系数。

```
## generate predictions and the average prediction
d.test$yhat1 <- predict(m1, newdata = d.test)
d.test$yhat2 <- predict(m2, newdata = d.test)
d.test$yhat3 <- predict(m3, newdata = d.test)
d.test$yhatavg <- rowMeans(d.test[, paste0("yhat", 1:3)])

## correlation in the testing data
cor(d.test)
```

	x	y	yhat1	yhat2	yhat3	yhatavg
x	1.000	0.44	1.000	-0.098	0.60	0.55
y	**0.442**	**1.00**	**0.442**	**0.753**	**0.87**	**0.91**
yhat1	1.000	0.44	1.000	-0.098	0.60	0.55
yhat2	-0.098	0.75	-0.098	1.000	0.69	0.76
yhat3	0.596	0.87	0.596	0.687	1.00	0.98
yhatavg	0.552	0.91	0.552	0.765	0.98	1.00

在这个结果中我们能看到，三个模型预测的平均确实比任何一个模型的个别表现都要好。然而，这只是在每个模型的表现差不多好的时候，才可以保证是真的。例如，我们假设一个病态的情况，其中一个模型完美地预测了结果而另一个的预测是和结果完全不相关的随机噪声。这样的话，两个模型的平均肯定会产生比只使用好的模型表现更坏。一般来说，检查被平均的模型至少在训练数据中有相似的性能是有好处的，理想的是在模型的预测之间有较低的相关性，因为这会产生最佳运行的平均。

集成方法是一种运用了模型平均的方法。一个常见的技术被称为自助聚集（bootstrap aggregating），其中数据采用替换抽样来形成相等规模的数据集，模型在每一个数据上训练，然后将这些结果平均。因为数据是用替换抽样的，在每个数据集中，某些样例会出现多次或者根本不会出现。因为模型是在每个数据集上训练的，如果某个特殊的变化对少数样例或数据的一个罕见巧合是独特的，它可能只在一个模型中出现。当预测在由每个重抽样的数据集所训练的许多模型上平均时，这种过拟合往往会减少。我们称这个过程为装袋（bagging）（自助聚集）。在某些背景下（例如决策树），会采用进一步的步骤来减少不同模型之间的相关性。例如，随机森林使用了自助聚集的决策树，但也在每个分划的节点上随机地选择了一个特征子集，目的是减少模型与模型的相关性并由此提高整体的平均性能。

装袋和模型平均在深度神经网络中并不常用，因为训练每个模型的成本会相当高，所以就时间和计算资源来说，多次重复这个过程会变得相当昂贵。然而，在 3.4 节中要讨论的丢弃过程，对许多子集模型训练的方式提供了相似的功能。尽管如此，

在深度学习的背景下使用模型平均还是可能的，即使它只能用在少数而非数百个模型上，就像在随机森林或某些其他方法中所常见的那样。

3.4 用例——使用丢弃提升样本外模型性能

丢弃是一种相对较新的正则化方法，对于大型和复杂的深度神经网络特别有价值。关于在深度神经网络中丢弃的更详细的探索，可以参见 Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. 和 Salakhutdinov, R. (2014)。丢弃背后的概念是相当直接的。在模型训练的过程中，单元（例如输入、隐藏神经元等）连同所有从它们出发的和到达它们的联系一起按照概率被丢弃了。例如，图 3-3 是一个模型训练中每一步会发生什么的例子，模型中的隐藏神经元和它们的连接是以 1/3 的概率来丢弃的。显示为灰色和虚线的神经元和连接就是那些被丢弃的。重要的是，某些神经元并不是在整个训练过程中被丢弃，而在某一步骤/更新中被丢弃。

考虑丢弃的一种方法是它会迫使模型对于扰动更加稳健。尽管许多神经元被包括在完整的模型中，但在训练期间它们并不总是同时存在的，所以某些神经元必须要操作的比它们不得已的其他方式要更自由一些。值得注意的是，输入也能和隐藏神经元一样被丢弃，但通常不会这样做，或者只做到较少的程度。

观察丢弃的另一种方式是，如果我们有一个大型的模型，隐藏神经元之间有 N 个权重，但有 50%会在训练期间丢弃，尽管所有的 N 个权重会在训练的某个阶段被用到，我们已经有效地把模型的复杂性减半，因为权重的平均个数将会减半。这减少了模型的复杂性，因此会有助于防止数据的过拟合。根据这个特征，如果丢弃的比例是 p，Srivastava, N., Hinton, G., Krizhevsky,A., Sutskever, I.和 Salakhutdinov, R. (2014)推荐按比例 $1/p$ 放大模型的复杂性，以便用一个大致相同复杂的模型来结束。

尽管神经元可以在训练期间随机地丢弃，但在测试期间，基于丢弃了一些神经元的模型计算许多预测，然后再平均来自每个模型的预测，这在计算上不方便。相反，已经有建议（而且似乎表现优异）说我们应该使用一个近似的平均，基于来自

单个神经网络的权重的归一化，而这个网络是基于每个权重被包括进来的概率的（即 1−p，尽管这可以通过经验而非理论来完成）。

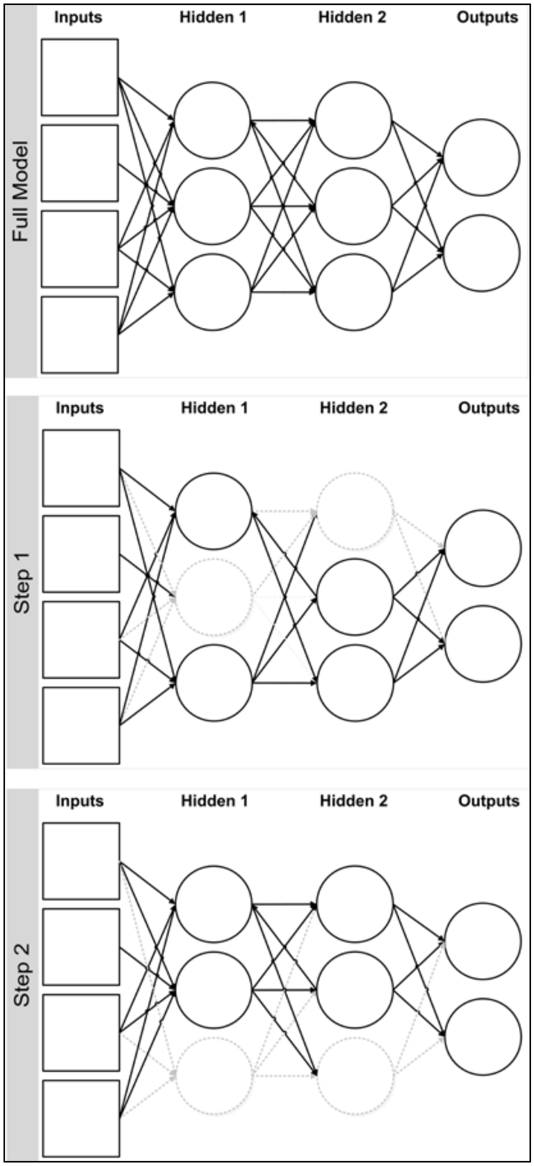


图 3-3

除了良好的表现之外，这种近似的权重校正是一种相当琐碎的计算。因此，丢弃的首要计算成本来自于这样一个事实，我们必须使用一个有着更多神经元和权重的模型，因为在每次训练更新过程中会丢弃许多的神经元和权重（对于隐藏神经元，推荐的值大约是 50%）。

尽管丢弃在计算上是相当廉价的，但它更慢，因为丢弃需要一个更大的模型，这种方法的一个潜在缺点就是，用更少的神经元和更快的学习率，某些权重会变得相当大。幸运的是，我们有可能将丢弃和其他形式的正则化比如 L1 和 L2 罚函数一起使用，放在一起考虑，结果是一个更大的模型，它能快速地（以一个更快的学习率）探索更广阔的参数空间，但为了使权重得到控制，它是通过丢弃和惩罚来正则化的。

为了显示丢弃在神经网络中的使用，我们返回到之前处理过的美国国家标准与技术研究所（Modified National Institute of Standards and Technology ，MNIST）数据集（我们在第 2 章中从 kaggle 下载过这个数据集）。我们将使用来自 deepnet 包的 nn.train() 函数，因为它允许丢弃。和第 2 章一样，我们并行地运行四个模型来减少它所用的时间。特别地，我们比较这四个模型，两个采用丢弃正则化、另两个不用，而且要么是 40 个要么是 80 个神经元。对于丢弃，我们分别为隐藏的和可见的单位指定丢弃的比例。基于经验法则，大约 50%的隐藏单位（和大约 80%的观察单位）应该保留，相应地，我们指定丢弃的比例为 0.5 和 0.2。

```
## Fit Models
nn.models <- foreach(i = 1:4, .combine = 'c') %dopar% {
set.seed(1234)
 list(nn.train(
    x = as.matrix(digits.X),
    y = model.matrix(~ 0 + digits.y),
    hidden = c(40, 80, 40, 80)[i],
    activationfun = "tanh",
    learningrate = 0.8,
    momentum = 0.5,
    numepochs = 150,
```

```
    output = "softmax",
    hidden_dropout = c(0, 0, .5, .5)[i],
    visible_dropout = c(0, 0, .2, .2)[i]))
}
```

接下来，我们可以循环模型，获得预测值并得到模型的整体性能。

```
nn.yhat <- lapply(nn.models, function(obj) {
  encodeClassLabels(nn.predict(obj, as.matrix(digits.X)))
})

perf.train <- do.call(cbind, lapply(nn.yhat, function(yhat) {
  caret::confusionMatrix(xtabs(~ I(yhat - 1) + digits.y))$overall
}))
colnames(perf.train) <- c("N40", "N80", "N40_Reg", "N80_Reg")

options(digits = 4)
perf.train
```

	N40	N80	N40_Reg	N80_Reg
Accuracy	**0.9050**	**0.9546**	**0.9212**	**0.9396**
Kappa	0.8944	0.9495	0.9124	0.9329
AccuracyLower	0.8965	0.9485	0.9134	0.9326
AccuracyUpper	0.9130	0.9602	0.9285	0.9460
AccuracyNull	0.1116	0.1116	0.1116	0.1116
AccuracyPValue	0.0000	0.0000	0.0000	0.0000
McnemarPValue	NaN	NaN	NaN	NaN

当我们在样本内数据中评价模型的时候，看起来 40 个神经元的有正则化的模型比没有正则化的模型表现要好，但是没有 80 个神经元的正则化的模型比有正则化的模型表现得要好。当然，真正的检验要在测试或留出数据上来做。

```
i2 <- 5001:10000
test.X <- digits.train[i2, -1]
test.y <- digits.train[i2, 1]
```

```
nn.yhat.test <- lapply(nn.models, function(obj) {
  encodeClassLabels(nn.predict(obj, as.matrix(test.X)))
})

perf.test <- do.call(cbind, lapply(nn.yhat.test, function(yhat) {
  caret::confusionMatrix(xtabs(~ I(yhat - 1) + test.y))$overall
}))
colnames(perf.test) <- c("N40", "N80", "N40_Reg", "N80_Reg")

perf.test
                  N40    N80  N40_Reg  N80_Reg
Accuracy       0.8652 0.8684   0.8868   0.9014
Kappa          0.8502 0.8537   0.8742   0.8904
AccuracyLower  0.8554 0.8587   0.8777   0.8928
AccuracyUpper  0.8746 0.8777   0.8955   0.9095
AccuracyNull   0.1074 0.1074   0.1074   0.1074
AccuracyPValue 0.0000 0.0000   0.0000   0.0000
McnemarPValue     NaN    NaN      NaN      NaN
```

测试数据相当好地突出了这个事实，在没有正则化的模型中，增加的神经元并不意味着提升了在测试数据上的模型性能。而且，样本内表现也过于乐观了（对于 80 个神经元非正则化的模型，在训练和测试数据中分别为，准确度=0.9546 和准确度=0.8684）。然而，这里我们看到了关于 40 个和 80 个神经元正则化模型共同的优点。尽管它们在测试数据中都比在训练数据中的性能要差，但在测试数据中，它们比对应的非正则化模型的性能要好。对于 80 个神经元的模型这个差异尤为重要，因为从训练数据到测试数据在总准确度上有 0.0862 的下降，但在正则化的模型中这个下降仅为 0.0382，导致了正则化的 80 个神经元的模型有最好的整体性能。

尽管这些数字绝不是创纪录的，但它们的确显示了丢弃，或者更通俗地说是正则化的价值，以及为了提升最终的测试性能该怎样处理尝试调节模型和丢弃参数。

3.5 小结

本章提供了几种方法来防止过拟合，包括常见的罚函数——L1 罚函数和 L2 罚函数、更简单模型的集成以及丢弃，其中变量和/或样例被去掉来使模型更杂乱并且防止了过拟合。我们检查了在回归问题和神经网络中罚函数的作用。在第 4 章，我们会进入深度学习和深度神经网络，看一看怎样把预测模型的准确度和表现推得更远。

第 4 章
识别异常数据

在本章中，我们将深入研究深度神经网络和深度学习模型。这一章将关注自动编码器，它可以用来学习数据集的特征。本章的第一部分将介绍无监督学习，这种方法中没有要预测的具体结果。接下来 4.2 节在机器学习特别是深度神经网络背景下，提供了自动编码器模型的一个概念综述。本章的主要核心将介绍如何建立并运用自动编码器模型识别异常的数据。这种非典型的数据可能仅仅是坏数据或者离群数据，但这种技术也能用于欺诈检测。例如，当一张个人信用卡的消费模型不同于通常的行为时，就可能是有些地方不对劲儿，要亮起红灯了。最后，本章以如何微调模型的一些探索来结束，包括使用第 3 章所讨论的不同正则化策略。除了深度学习模型自身的用处，本章还将提供使用和训练它们的重要组件。

这一章将会包括下列主题。

- 什么是无监督学习
- 自动编码器如何工作
- 在 R 中训练自动编码器
- 用例——建立并训练自动编码器模型

- 微调自动编码器

4.1 无监督学习入门

到目前为止，我们已经关注过大体上可以归入有监督学习类型的模型和技术。有监督学习是有监督的，意思是这种任务是机器要取学习一组变量或特征与一个或多个结果之间的关系，通常只有单个结果。例如，一家公司也许想要预测某人是否有可能成为它的客户，在这种情形下，一个人是否会成为客户的结果被编码为是/否。在本章中，我们将深入研究无监督学习的方法。与使用了一个结果变量或者标记数据的有监督学习不同，无监督学习只使用输入的特征来学习。一个常见的无监督学习的例子是聚类分析，机器去学习数据中隐藏的或者潜在的聚类，目标是去最小化评价的准则（例如一个聚类内部的最小的方差）。

另一种关于无监督学习的方法是，它的目标是去预测输入值。这种方法的一个例子如图 4-1 所示，我们刚开始看这是有点反直觉的，因为去学习一个目标仅仅是重复产生输送给它的输入值的复杂模型，这看起来相对而言没有什么用。然而，这里有一些有用的特征。一种常用的无监督学习是降维。降维的目标是对一组 p 个变量找到一组隐藏的 k 个变量，且 $k<p$，但是用这 k 个隐变量能够合理地重现 p 个原始变量。这通常是一种取舍和平衡的行为，因为通常降维越厉害，简化就越厉害，但要以精度为代价。

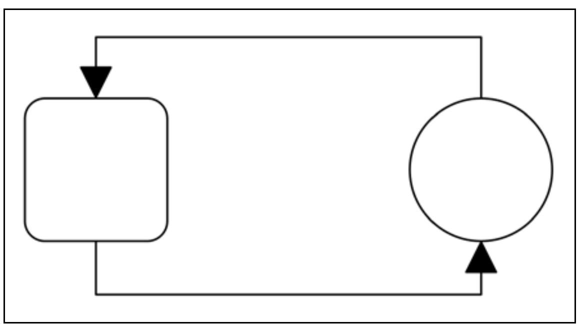


图 4-1

或许，最常见的降维的例子是主成分分析。主成分分析使用一个正交变换把原始数据变为主成分。除了彼此不相关之外，主成分还按照从解释最多方差的成分到解释最少方差的成分来排序。虽然所有的主成分都是可以使用的（这种情况下数据的维度没有减少），但是只有那些解释了足够大数量方差（例如基于高的特征值）的成分才会被引入模型，而那些考虑了相对较少方差的成分会作为噪声或者不必要的成分而被舍弃。

4.2 自动编码器如何工作

与到目前为止我们已经讨论过的其他神经网络一样，自动编码器也是神经网络，它可以是浅层的也可以是深度的。自动编码器和其他的神经网络的区别是自动编码器是被训练用来重新生成输入值或者预测输入值的。因此隐藏层和神经元并不是在一个输入值和某些其他结果之间映射，而是自（自动）编码的。

与那些更常见的神经网络的情况不同，它们的结果是一些我们想要去预测的变量。给定足够的复杂性，自动编码器只能学习恒等函数，而且隐藏神经元将会精确地反映原始数据，产生没有意义的收获。因为用于训练的结果和输入值是相同的，最好的自动编码器并不必需是最准确的那一个，而是能揭示数据中一些有意义的结构或者架构的那一个，或者是减少噪声、识别离群点或异常数据、有着某些作用但并不必然涉及模型输入的精确预测的附带作用的那一个。

使用自动编码器的一种方法是执行降维。比原始数据有更低维度的自动编码器叫作亚完备的（undercomplete）。利用亚完备的自动编码器，我们能使自动编码器学习最显著或最突出的数据特征。于是，这些新的隐藏特征能用于进一步分析或处理。例如，一个自动编码器重要而且常见的应用是去预先训练深度神经网络或者其他的有监督模型。除此之外，直接解释隐藏特征本身也许是有可能的而且也是有趣的。例如，它们可以提供关于数据中的关键特征或者结构的深入理解。

使用亚完备模型是一种正则化模型的有效方法。然而，如果使用某些其他方式

的正则化，也有可能训练出过完备（overcomplete）的自动编码器，其中隐藏的维数大于原始数据。接下来我们将更深入地讨论正则化的不同形式。

和规则的神经网络一样，对于自动编码器也有两个部分。首先，一个编码函数 $f(\cdot)$，把原始数据编码传递给隐藏神经元；其次，一个解码函数 $g(\cdot)$，将解码回原始数据。

正则化的自动编码器

一个亚完备的自动编码器，在某种程度上是一种正则化的自动编码器的形式，其中的正则化是通过使用一个比数据更浅的（或者换句话说，更低的）维度表示来发生的。但是，正则化也可以通过其他含义来实现。

1．惩罚的自动编码器

如同我们在第 3 章中所见，一种方法是使用惩罚。通常，我们的目的是（尽可能简单地）最小化重构误差。如果有一个目标函数，传统上，我们可以优化 $F(y, f(x))$，其中 $f(\cdot)$编码原始数据的输入来生成预测的或者期望的 y 值。对于自动编码器来说，我们有 $F(x, g(f(x)))$，所以机器来学习权重以及 $f(\cdot)$和 $g(\cdot)$的函数形式，最小化 x 与 x 的命名为 $g(f(x))$的重构物之间的不相符合之处。如果想要一个过完备的自动编码器，我们需要引入某种程度的正则化来强迫机器去学习一种表示法，这种表示法并非仅仅是输入值的完全镜像。例如，我们可能会加入一个基于复杂性来进行惩罚的函数，这样，我们不优化 $F(x, g(f(x)))$，而是优化 $F(x, g(f(x))) + P(f(x))$，其中的罚函数 P 依赖于编码或者原始输入 $f(\cdot)$，这样的惩罚在某种程度上不同于我们之前见过的惩罚形式，然而，那里的惩罚设计不是为了减少参数的稀疏性，而是为了减少原始数据编码表达的潜变量 H 的稀疏性。其学习目标是捕捉数据本质特征的潜变量形式。

可以用来提供正则化的惩罚的另一种类型是基于导数的方法。不同于稀疏的自编码器具有减少潜变量的稀疏性的惩罚，惩罚模型中的求导结果，学习了一种对原始输入数据 x 的小扰动相对不敏感的 $f(x)$，更确切地说，它对 x 中的改变编码变化

很大的函数施加了惩罚，选择了那些梯度相对平坦的区域。

2．去噪自动编码器

去噪自动编码器（denoising auto-encoders）移除了噪声或者对数据去噪，而且是学习原始数据中的潜在表示（Vincent, P., Larochelle, H., Bengio, Y.和 Manzagol, P. A. (2008, July); Bengio, Y., Courville, A.和 Vincent, P. (2013)）的一种有用的技术。我们说过自动编码器的一般任务是去优化 $F(x,g(f(x)))$。然而对于一个去噪自动编码器来说，是从有噪声的或者记为 $\underset{\sim}{x}$ 的 x 损坏版本中去恢复 x。所以，任务就变成了优化 $F(x,g(f(\underset{\sim}{x})))$ 。

尽管去噪编码器是用来尝试从损坏数据或者有噪声的数据中恢复“真正的”表示，这项技术也可以用作一种正则化的工具。作为一种正则化方法，原始数据是故意损坏的，而不是有噪声的或者损坏的数据并且试图恢复真实。这促使自动编码器要做的比仅仅去学习恒等函数要多得多，因为原始输入（ $\underset{\sim}{x}$ ）不再和输出 x 相同。这个过程如图 4-2 所示。

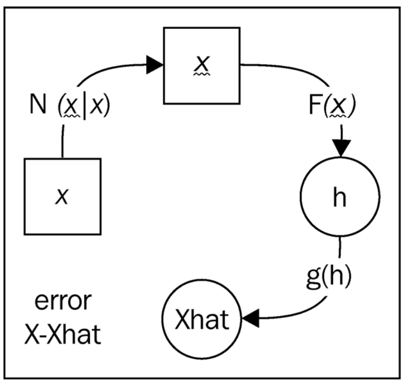


图 4-2

剩下的是选择什么样的函数 $N(\bullet)$，这个函数应该增加 x 的噪声或者损坏。这里有两种增加噪声的选择，通过一个随机过程或者对任何给定的训练循环中包括进来一个原始 x 输入的子集。接下来，我们来探索如何在 R 中真正地训练一个自动编码器模型。

4.3 在 R 中训练自动编码器

为了训练第一个自动编码器，我们首先需要 R 做好准备。除了在 `checkpoint.R` 中的其他的 R 包，我们还要添加 data.table 包来帮助数据管理，如下列代码所示。

```
library(data.table)
```

现在我们可以得到 `checkpoint.R` 文件来建立用于分析的 R 环境，如下所示。

```
source("checkpoint.R")
options(width = 70, digits = 2)
```

对于这些最初的例子，我们将使用美国国家标准和技术研究所（Modified National Institute of Standards and Technology，MNIST）的数字图像数据。与之前的章节一样，下面的代码载入所需的数据并且建立用于分析的 H2O 集群。我们使用数据的前 20 000 行来训练并将接下来的 10 000 行用于测试。除了载入数据并建立 H2O 集群，我们需要将数据转移到 H2O，这由 `as.h2o()` 函数来完成。

```
## data and H2O setup
digits.train <- read.csv("train.csv")
digits.train$label <- factor(digits.train$label, levels = 0:9)

cl <- h2o.init(
  max_mem_size = "20G",
  nthreads = 10)

h2odigits <- as.h2o(
  digits.train,
  destination_frame = "h2odigits")
```

```
i <- 1:20000
h2odigits.train <- h2odigits[i, -1]

itest <- 20001:30000
h2odigits.test <- h2odigits[itest, -1]
xnames <- colnames(h2odigits.train)
```

为了便于分析，我们使用 `h2o.deeplearning()` 函数，这个函数有很多选项，而且提供了 H2O 中可用的所有深度学习特征。在我们着手如何编写模型代码时候，对可重复性做一个简要的评论是符合规程的。通常，为了使运行代码的结果完全可重复，我们可能要设置随机数种子。H2O 使用一种称为 Hogwild！的并行方法，这种方法并行了随机梯度下降优化，用于如何优化/决定模型权重（参见 Niu, F., Recht, B., Ré, C.,和 Wright, S. J. (2011)的 Hogwild!:A Lock-Free Approach to Parallelizing Stochastic Gradient Descent，https://www.eecs.berkeley.edu/~brecht/papers/hogwildTR.pdf）。因为 Hogwild!起作用的方式，使得结果完全可重复是不太可能的，所以，当我们运行这些代码时，或许会得到稍微不同的结果。

在 `h2o.deeplearning()` 函数的调用中，第一个参数是 *x* 或者输入变量名的列表。训练框（training frame）是 H2O 数据用于模型训练的，验证框（validation frame）只是在没用来训练的数据上评价模型性能。接下来，我们指定用在这里的激活函数——`Tanh`，在接下来的有关深度学习预测的章节中我们将会讨论这个函数更多的细节。通过设置参数 `autoencoder = TRUE`，这个模型就成为一个自动编码器模型，而不是规则模型，所以没有需要指定的 *y* 或者输出变量。

尽管使用了一个深度学习函数，为了入门我们仍要用一个单层（浅层）的隐藏神经元。这里有 20 次的训练迭代，叫作轮数（epoch）。其余的参数只是对这个模型指定不使用任何正则化。正则化是不需要的，因为这里有数百个输入变量但只有 50 个隐藏神经元，所以相对简单的模型提供了所有需要的正则化。最后，所有的结果存储在一个 R 对象 `m1` 里。

```
m1 <- h2o.deeplearning(
  x = xnames,
```

```
  training_frame= h2odigits.train,
  validation_frame = h2odigits.test,
  activation = "Tanh",
  autoencoder = TRUE,
  hidden = c(50),
  epochs = 20,
  sparsity_beta = 0,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(0),
  l1 = 0,
  l2 = 0
)
```

其余的模型和第一个模型相似，但是通过增加隐藏神经元的个数和加入正则化，我们调整了模型的复杂性。具体来说，模型 m2a 没有任何正则化，但是把隐藏神经元的个数增加到了 100 个。模型 m2b 使用 100 个隐藏神经元还有一个.5 的稀疏 beta 值。最后，模型 m2c 使用 100 个神经元和输入（*x* 变量）的 20%的丢弃，这会导致一种损坏的输入形式，所以模型 m2c 是一种去噪自动编码器的形式。

```
m2a <- h2o.deeplearning(
  x = xnames,
  training_frame= h2odigits.train,
  validation_frame = h2odigits.test,
  activation = "Tanh",
  autoencoder = TRUE,
  hidden = c(100),
  epochs = 20,
  sparsity_beta = 0,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(0),
  l1 = 0,
  l2 = 0
)

m2b <- h2o.deeplearning(
```

```
    x = xnames,
    training_frame= h2odigits.train,
    validation_frame = h2odigits.test,
    activation = "Tanh",
    autoencoder = TRUE,
    hidden = c(100),
    epochs = 20,
    sparsity_beta = .5,
    input_dropout_ratio = 0,
    hidden_dropout_ratios = c(0),
    l1 = 0,
    l2 = 0
  )

  m2c <- h2o.deeplearning(
    x = xnames,
    training_frame= h2odigits.train,
    validation_frame = h2odigits.test,
    activation = "Tanh",
    autoencoder = TRUE,
    hidden = c(100),
    epochs = 20,
    sparsity_beta = 0,
    input_dropout_ratio = .2,
    hidden_dropout_ratios = c(0),
    l1 = 0,
    l2 = 0
  )
```

根据在 R 中键入存储的模型对象的名字，我们会得到模型和它的性能的汇总。为了节省空间，许多输出被省略了，但是对每一个模型，我们在后面的输出中用训练数据和验证数据中的均方误差（mean squared error，MSE）作为性能的表示。取值为零的 MSE 表明了一个完美的拟合，更高取值的 MSE 表明了在 $g(f(x))$和 x 之间的偏差。

在模型 m1 中，MSE 是相当低的，而且在训练数据和验证数据中是完全相同的。这或许部分归因于这是一个相对简单的模型（50 个隐藏神经元和 20 个轮数，同时有数百个输入变量）。在模型 m2a 中，在 MSE 上有 45%的减少，尽管两者都很低。但是伴随着更大的模型复杂性，我们可以观察到在训练和验证度量中轻微的差异。在 m2b 中我们也注意到相似的结果。尽管事实上验证度量没有随着正则化而提升，也暗示正则化的训练数据的性能泛化得更好。在 m2c 中，没有额外模型复杂性的 20%的输入丢弃在训练数据和验证数据中都导致了更差的性能。100 个隐藏神经元的初始模型太简单了，真的不需要很多的正则化。

```
m1
Training Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.014

H2OAutoEncoderMetrics: deeplearning
** Reported on validation data. **

Validation Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.014

m2a
Training Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.0076

H2OAutoEncoderMetrics: deeplearning
** Reported on validation data. **

Validation Set Metrics:
=====================
```

```
MSE: (Extract with 'h2o.mse') 0.0079
```

m2b

```
Training Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.0077

H2OAutoEncoderMetrics: deeplearning
** Reported on validation data. **

Validation Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.0079
```

m2c

```
Training Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.0095

H2OAutoEncoderMetrics: deeplearning
** Reported on validation data. **

Validation Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.0098
```

我们能审视模型结果的另一种方法是计算这个模型的异常程度，可以使用 h2o.anomaly() 函数来完成。这个结果被转换为数据框，做了标记并且合并到一个叫作“error”的最后的数据表格对象中。

```
error1 <- as.data.frame(h2o.anomaly(m1, h2odigits.train))
error2a <- as.data.frame(h2o.anomaly(m2a, h2odigits.train))
error2b <- as.data.frame(h2o.anomaly(m2b, h2odigits.train))
error2c <- as.data.frame(h2o.anomaly(m2c, h2odigits.train))

error <- as.data.table(rbind(
  cbind.data.frame(Model = 1, error1),
  cbind.data.frame(Model = "2a", error2a),
  cbind.data.frame(Model = "2b", error2b),
  cbind.data.frame(Model = "2c", error2c)))
```

接下来，我们会使用 data.table 包创建一个新的数据对象 `percentile`，对于每个模型这个对象包含了 99%分位数。

```
percentile <- error[, .(
  Percentile = quantile(Reconstruction.MSE, probs = .99)
), by = Model]
```

将模型得到的每种情况的异常程度以及 99%分位数的信息组合起来，我们能用 ggplot2 包画出结果。直方图显示了每一种情况的误差比，虚线是 99%分位数。任何超过 99%分位数的值都可以认为是相当极端或者异常的。

```
p <- ggplot(error, aes(Reconstruction.MSE)) +
  geom_histogram(binwidth = .001, fill = "grey50") +
  geom_vline(aes(xintercept = Percentile), data = percentile,
linetype =2) + theme_bw() + facet_wrap(~Model)
print(p)
```

这个结果如图 4-3 所示。模型 2a 和 2b 有最低的误差比，而且我们能看到小的尾部。

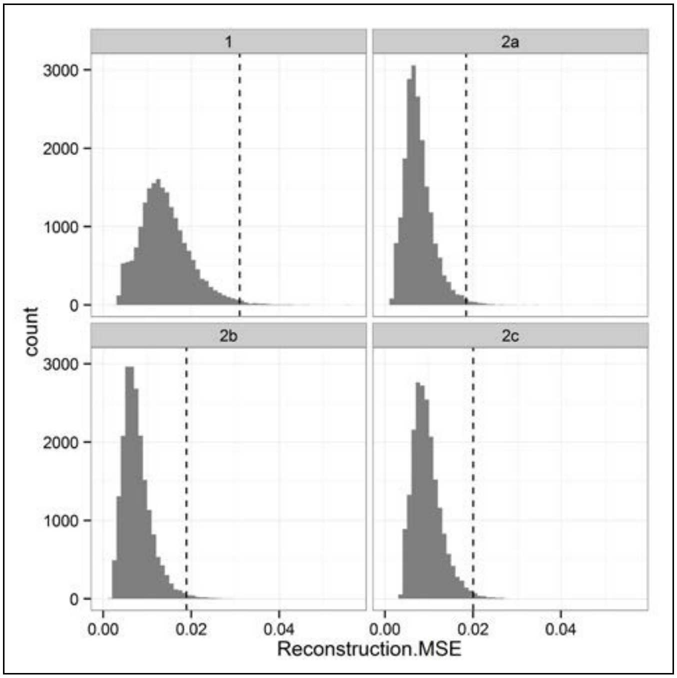


图 4-3

如果我们横向合并数据，每个模型的异常值在不同的列当中而不是在另一个指示了模型的很长的列中，我们能够针对两两变量画出异常值。结果如图 4-4 所示，而且其中显示了模型之间更高的对应程度，往往伴随着对一个模型异常对另一个模型也异常的情况。

```
error.tmp <- cbind(error1, error2a, error2b, error2c)
colnames(error.tmp) <- c("M1", "M2a", "M2b", "M2c")
plot(error.tmp)
```

我们能检查模型结果的另一种方法是提取模型的深度特征。（一层层的）深度特征可以使用 `h2o.deepfeatures()` 函数来提取。深度特征是模型中隐藏神经元的值。探索这些特征的一种方法是计算它们的相关性并检查相关系数的分布，如下所示代码，再一次使用 ggplot2 包。结果如图 4-5 所示。一般来讲，深度特征有小的相关系数 r，绝对值小于.20，只有极少数|*r*|>.20。

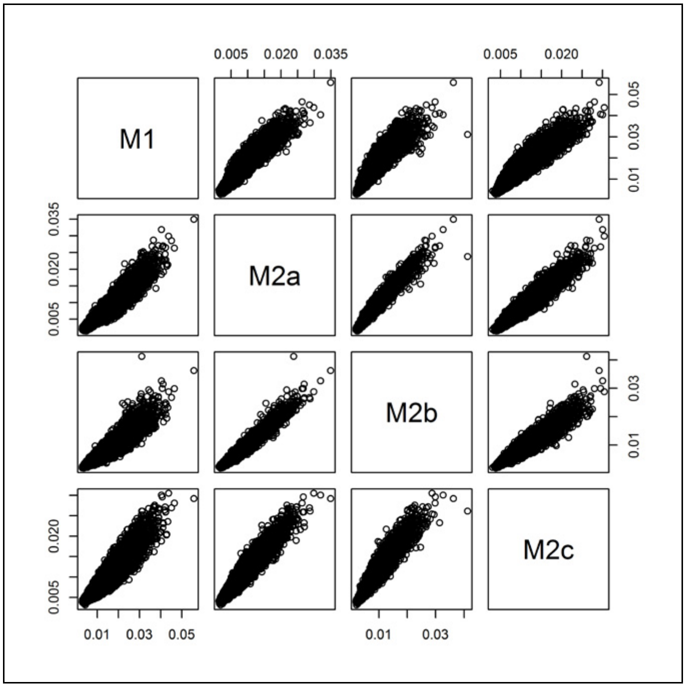

M1
M2a
M2b
M2c
0.005 0.020 0.035
0.005 0.020
0.01 0.03 0.05
0.01 0.03 0.05
0.005 0.020 0.035
0.01 0.03
0.005 0.020
0.01 0.03

图 4-4

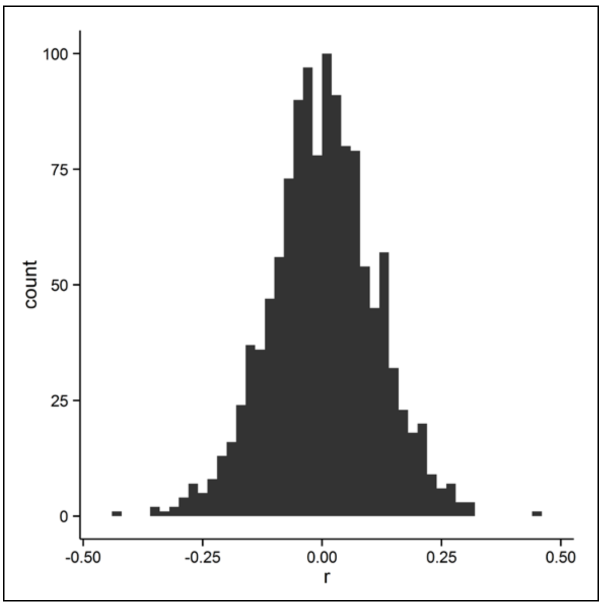

count
100
75
50
25
0
-0.50 -0.25 0.00 0.25 0.50
r

图 4-5

```
features1 <- as.data.frame(h2o.deepfeatures(m1, h2odigits.train
))
r.features1 <- cor(features1)
r.features1 <- data.frame(r = r.features1[upper.tri(r.features1
)])

p.hist <- ggplot(r.features1, aes(r)) +
  geom_histogram(binwidth = .02) +
  theme_classic()
print(p.hist)
```

到目前为止例子显示了自动编码器是如何训练的，但是只表示了有一个隐藏层的浅层自动编码器。我们也可以有多个隐藏层的深度自动编码器。

考虑到我们知道 MNIST 数据集包含 10 个不同的手写体数字，或许我们可以尝试增加只有 10 个神经元的第二层隐藏神经元，假设当模型学习数字特征的时候，10 个突出的特征会对应于这 10 个数字。

为了加上这第二层的隐藏神经元，我们为 `hidden` 参数传递一个向量 `c(100, 10)`，同时更新 `hidden_dropout_ratios` 参数，因为可以对每一个隐藏层使用不同的丢弃率。

```
m3 <- h2o.deeplearning(
  x = xnames,
  training_frame= h2odigits.train,
  validation_frame = h2odigits.test,
  activation = "Tanh",
  autoencoder = TRUE,
  hidden = c(100, 10),
  epochs = 30,
  sparsity_beta = 0,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(0, 0),
  l1 = 0,
```

```
  l2 = 0
)
```

和之前见到的一样，我们能对隐藏神经元提取值。这里我们再次使用 h2o.deepfeatures()函数，但是可以对于第 2 层指定我们想要的值。接下来显示的是这些特征的前六行。

```
features3 <- as.data.frame(h2o.deepfeatures(m3, h2odigits.train,
2))
head(features3)

 DF.L2.C1 DF.L2.C2 DF.L2.C3 DF.L2.C4 DF.L2.C5 DF.L2.C6  DF.L2.C7
1   -0.16     0.01     0.61    0.610   0.7468     0.11   -0.3927
2   -0.28    -0.77    -0.82    0.563  -0.4422    -0.66    0.6042
3   -0.48    -0.23     0.24   -0.141   0.3252     0.42   -0.0088
4   -0.30    -0.37     0.42   -0.313   0.1896    -0.27    0.1442
5   -0.36    -0.73    -0.84    0.733  -0.4807    -0.62    0.6828
6   -0.24     0.16    -0.10   -0.037  -0.0064    -0.20    0.4794
  DF.L2.C8 DF.L2.C9 DF.L2.C10
1    0.023    -0.39     0.385
2    0.321    -0.39    -0.079
3    0.589     0.59     0.538
4   -0.224    -0.31     0.557
5    0.347    -0.62    -0.098
6   -0.592     0.11     0.253
```

因为这里没有要预测的结果，这些值是连续的而并不作为某个特别数字的概率，仅仅是 10 个连续隐藏神经元的值。

接下来我们能加上来自训练数据的真实的数据标签，而且使用 melt()函数把数据重塑为一个长的数据集。从那里出发，根据数字实际上属于哪一种情况，我们能画出 10 个隐藏神经元每一个的均值。如果 10 个隐藏特征粗略地对应着 10 个数字，对于特别的标签（例如 0、3 等），它们应该会在深度特征上具有一个极端值，表明深度特征和真实数字之间的对应。结果如图 4-6 所示。

```
features3$label <- digits.train$label[i]
features3 <- melt(features3, id.vars = "label")

p.line <- ggplot(features3, aes(as.numeric(variable), value,
                      colour = label, linetype = label)) +
  stat_summary(fun.y = mean, geom = "line") +
  scale_x_continuous("Deep Features", breaks = 1:10) +
  theme_classic() +
  theme(legend.position = "bottom", legend.key.width = unit(1,
"cm"))print(p.line)
```

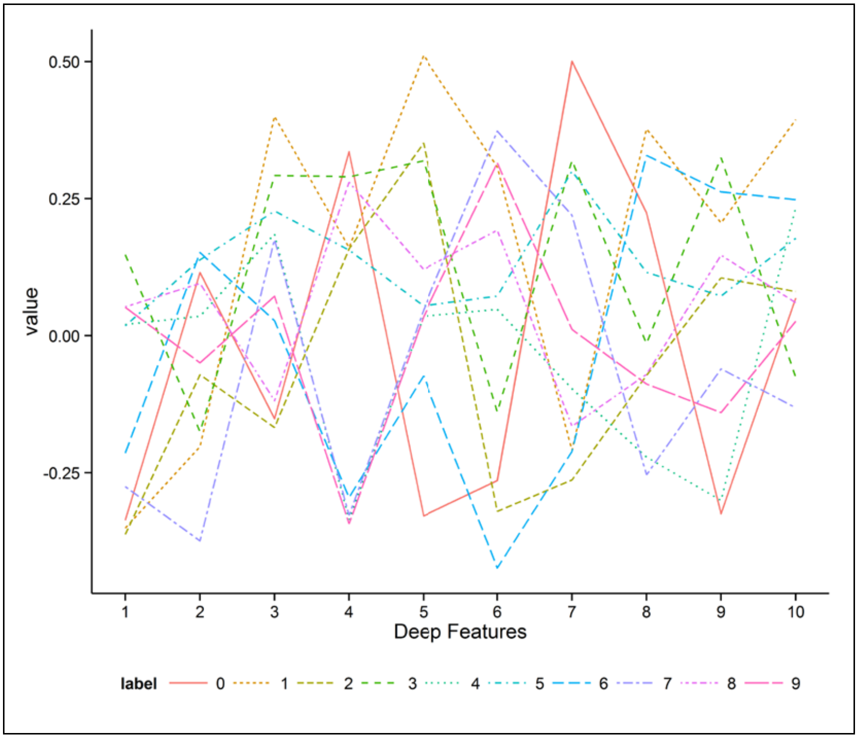


图 4-6

虽然这里看起来存在某些对应关系(例如,零在深度特征 4 和 7 上是特别高的),但是总的来说，结果是非常杂乱的，在深度特征和真实数字标签之间没有高区分度的特别明显的标示。

最后，我们可以看一下模型的性能度量。MSE 大约是 0.039，比起浅层模型来，深度模型的拟合相当差，这可能是因为在第 2 层中只有 10 个隐藏神经元，这太简单了，不足以捕捉为重建原始输入所需要数据的所有不同特征。

```
m3
Training Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.039

H2OAutoEncoderMetrics: deeplearning
** Reported on validation data. **

Validation Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.04
```

这一节已经介绍了训练自动编码器的模型、代码以及评价模型性能的一些方法的基础。接下来，我们来看一个用例：使用自动编码器发现异常值。

4.4 用例——建立并运用自动编码器模型

关于用例，我们使用之前验证过的来自智能手机的活动记录仪数据。这些数据包括活动记录仪记录的一些人在坐、站、卧、行走和上下楼梯时的数据。我们的目的是识别数据中的任何异常值或者那些反常或不同寻常的值。

首先，我们将把训练和测试数据载入到 R 中，然后把它转化到 H2O 中用于分析。

```
use.train.x <- read.table("UCI HAR Dataset/train/X_train.txt")
use.test.x <- read.table("UCI HAR Dataset/test/X_test.txt")
```

```
use.train.y <- read.table("UCI HAR Dataset/train/y_train.txt")
[[1]]
use.test.y <- read.table("UCI HAR Dataset/test/y_test.txt")[[1
]]

use.labels <- read.table("UCI HAR Dataset/activity_labels.txt")

h2oactivity.train <- as.h2o(
  use.train.x,
  destination_frame = "h2oactivitytrain")

h2oactivity.test <- as.h2o(
  use.test.x,
  destination_frame = "h2oactivitytest")
```

随着数据载入，我们做好准备去训练模型。模型的建立过程和我们曾经训练过的初始模型极为相似。在这里我们使用两个层，每层有 100 个神经元。目前，这里没有使用特定的正则化，当然，隐藏神经元比起输入的变量明显要少，模型的简单性可以提供足够的正则化。

```
mu1 <- h2o.deeplearning(
  x = colnames(h2oactivity.train),
  training_frame= h2oactivity.train,
  validation_frame = h2oactivity.test,
  activation = "Tanh",
  autoencoder = TRUE,
  hidden = c(100, 100),
  epochs = 30,
  sparsity_beta = 0,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(0, 0),
  l1 = 0,
  l2 = 0
)
```

检查模型的性能，它有非常低的重建误差。这说明为了捕捉数据中的关键特征，模型足够复杂。训练数据和测试数据之间在模型性能中没有重大的差异。

```
mu1
Training Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.001

H2OAutoEncoderMetrics: deeplearning
** Reported on validation data. **

Validation Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.0011
```

我们能提取出每种情况的异常程度并且画出分布。结果如图 4-7 所示。显而易见这里有一些情况比其余的情况更加异常，因为显示出了更高的误差率。

```
erroru1 <- as.data.frame(h2o.anomaly(mu1, h2oactivity.train))

pue1 <- ggplot(erroru1, aes(Reconstruction.MSE)) +
  geom_histogram(binwidth = .001, fill = "grey50") +
  geom_vline(xintercept = quantile(erroru1[[1]], probs = .99),
linetype = 2) +
  theme_bw()
print(pue1)
```

试图更深入地探索这些异常的一种方法是，检查是否存在某种活动趋向于有更多或者更少的异常值。通过找到哪种情况异常，我们就能做到这一点，这里把异常情况定义为前 1%的误差率，然后提取那些情况的活动并且画出它们。这样做的结果如图 4-8 所示。绝大多数的异常情况来自于下楼或者躺下时。伴随着再现的输入中的高误差，深度特征或许对这些情况是一种（相对）差的输入表示。在实践中，

如果根据这些结果来分类，我们可能会排除这些情况，因为它们似乎不能拟合模型学习到的特征。

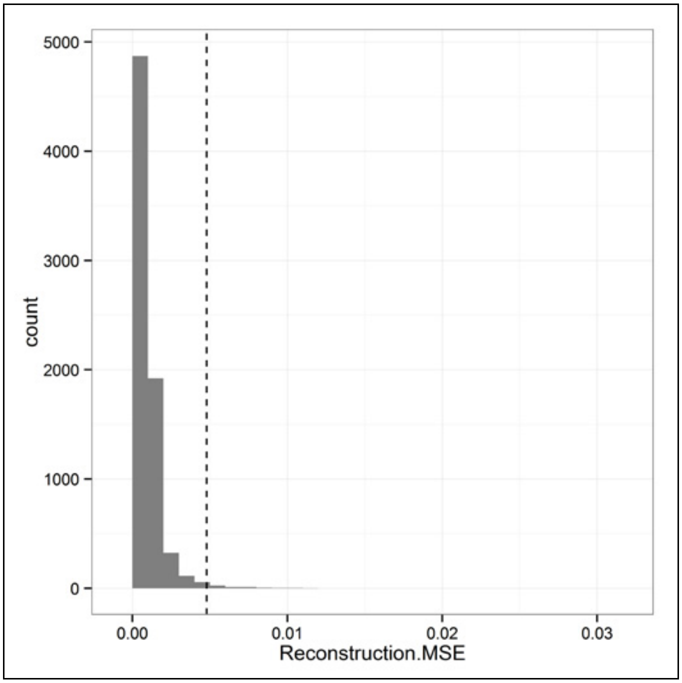


图 4-7

```
i.anomolous <- erroru1$Reconstruction.MSE >= quantile(erroru1[
[1]], probs = .99)

pu.anomolous <- ggplot(as.data.frame(table(use.labels$V2[use.
train.y[i.anomolous]])),
         aes(Var1, Freq)) +
    geom_bar(stat = "identity") +
    xlab("") + ylab("Frequency") +
    theme_classic() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1, vjus
t = 1))

# print the ggplot2 plot object
```

```
print(pu.anomolous)
```

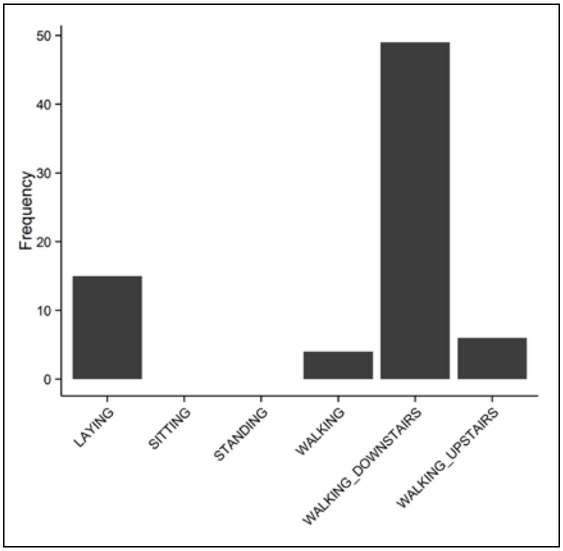


图 4-8

在这个例子中，我们使用深度自动编码器模型学习来自智能手机的活动记录仪数据的特征。这种方法对于排除未知的或不寻常的活动是很有用的，而不对它们进行不正确地分类。例如，作为一个要分类我们所从事的活动花费时间的 app 的一部分，与其在我们下楼梯的时候，去异常的调用一种行走或者坐下的活动，还不如在模型不确定或者隐藏特征没有充分重建输入值的地方离开几分钟，这或许更好。

这样的工作也有助于识别模型通常会在哪里出现更多的问题。或许我们需要更多的传感器和额外的数据去更多地表示下楼梯的情况，才能够理解为什么下楼梯会产生相对高的误差率。

这些深度自动编码器在其他重要的识别异常的场景中也很有用，比如说金融数据或者信用卡的使用模式。异常的消费模式或许表明有欺诈发生或者这张信用卡被盗了。与其试图在几百万次信用卡交易中人工搜索，不如训练一个自动编码器模型并用它来识别异常用于进一步地探索。

4.5 微调自动编码器模型

在本章的前面，我们已经学习了如何训练并且使用自动编码器模型。最后本节我们探索如何优化和微调自动编码器模型，去验证例如如何选取隐藏神经元的个数或者层数的这样问题。

有时候，或许是出于概念的原因，我们要假定关于数据的确定结构。然而，如果不是这样的情况，我们可以变化参数的值来获得最好的模型。在我们尝试多个模型并且要选择一个最佳模型的时候加重了一种困境，那就是，即使几个模型是等价的，因为偶然，在一个给定的样本内某一个模型的性能可以优于其他的模型。要解决这一点，从而只在使用训练数据的时候优化参数值，我们可以在训练过程中使用诸如交叉验证这样的技术，只是这个最终的模型需要使用留出数据或测试数据来验证。目前，对于自动编码器模型，H2O 还不支持交叉验证。如果我们真的想用交叉验证，可以手动来实现。使用来自 caret 包的 `createFolds()` 函数，我们能容易地实现这一点。

```
## create 5 folds
folds <- createFolds(1:20000, k = 5)
```

接下来，我们创建一个想要尝试微调的超参数列表，如下列的代码所示。

```
## create parameters to try
hyperparams <- list(
  list(
    hidden = c(50),
    input_dr = c(0),
    hidden_dr = c(0)),
  list(
    hidden = c(200),
    input_dr = c(.2),
```

```
    hidden_dr = c(0)),
  list(
    hidden = c(400),
    input_dr = c(.2),
    hidden_dr = c(0)),
  list(
    hidden = c(400),
    input_dr = c(.2),
    hidden_dr = c(.5)),
  list(
    hidden = c(400, 200),
    input_dr = c(.2),
    hidden_dr = c(.25, .25)),
  list(
    hidden = c(400, 200),
    input_dr = c(.2),
    hidden_dr = c(.5, .25)))
```

最后，我们对这些超参数循环一个 5 折交叉验证来训练所有的模型。因为我们正在训练 6×5 共 30 个模型，有些带有数百个隐藏神经元（注意，对这个模型要用增加的速度来运行，我们把 H2O 集群改变到一个 5 核 12G 内存的机器上），所以这要花上一些时间来完成。

```
fm <- lapply(hyperparams, function(v) {
  lapply(folds, function(i) {
  h2o.deeplearning(
    x = xnames,
    training_frame = h2odigits.train[-i, ],
    validation_frame = h2odigits.train[i, ],
    activation = "Tanh",
    autoencoder = TRUE,
    hidden = v$hidden,
    epochs = 30,
    sparsity_beta = 0,
    input_dropout_ratio = v$input_dr,
```

```
    hidden_dropout_ratios = v$hidden_dr,
    l1 = 0,
    l2 = 0
  )
  })
})
```

接下来，我们对结果进行循环并对验证数据提取 MSE，这里是没有用在交叉验证中的单个折。

```
fm.res <- lapply(fm, function(m) {
  sapply(m, h2o.mse, valid = TRUE)
})
```

我们把结果合并到一个数据表格里，观察并画出各折交叉验证的性能。

```
fm.res <- data.table(
  Model = rep(paste0("M", 1:6), each = 5),
  MSE = unlist(fm.res))

head(fm.res)
    Model         MSE
1:     M1 0.014619734
2:     M1 0.014655749
3:     M1 0.014651761
4:     M1 0.014310286
5:     M1 0.014303792
6:     M2 0.006781414
```

最后，我们做出结果的箱型图，来看它们是如何扩散的，或者是否有哪一个交叉验证的运行特别反常。结果如图 4-9 所示，似乎交叉验证中每一折的 MSE 都是相当接近的，这样均值/中位数就是相当合理的数据汇总。

```
p.erate <- ggplot(fm.res, aes(Model, MSE)) +
  geom_boxplot() +
```

```
  stat_summary(fun.y = mean, geom = "point", colour = "red") +
  theme_classic()
print(p.erate)
```

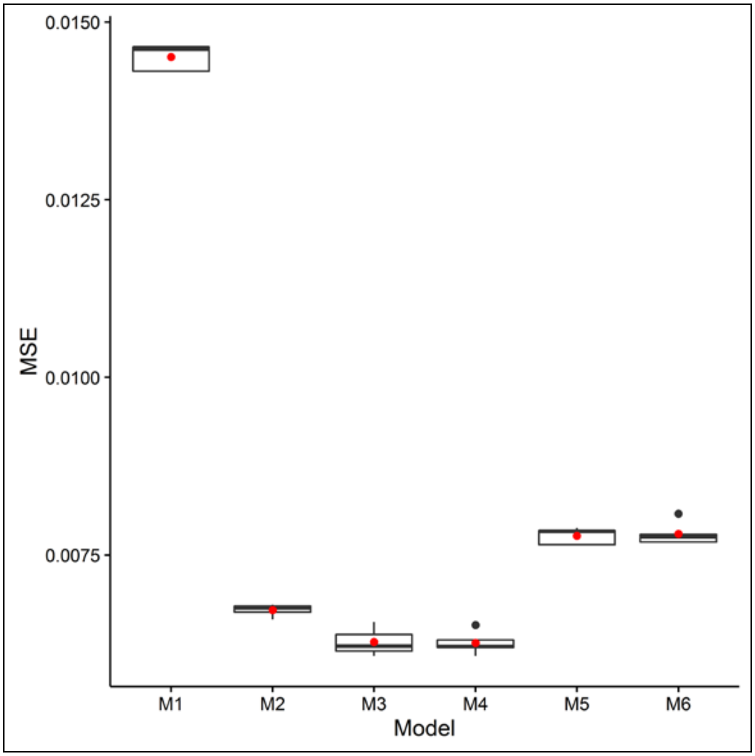


图 4-9

计算每个模型的平均 MSE 并从最小到最大进行排序，下面是我们得到的结果。

```
fm.res[, .(Mean_MSE = mean(MSE)), by = Model][order(Mean_MSE)]
    Model    Mean_MSE
1:     M4 0.006261764
2:     M3 0.006276417
3:     M2 0.006725956
4:     M5 0.007768764
5:     M6 0.007797575
6:     M1 0.014508264
```

看起来第四组超参数提供了最低的交叉验证 MSE。第四组超参数是一个相当复杂的模型，有 400 个隐藏神经元，但是也采用了 20%输入变量丢弃以及在每一次迭代中 50%隐藏神经元丢弃的正则化，而且这一组真正优于（虽然只是轻微的）有着相同模型复杂性但没有任何隐藏层丢弃的第三组超参数。尽管没有差太多，有着第二层 200 个隐藏神经元的深度模型的性能差于浅层模型。

选好了最佳模型，使用第四组超参数，我们能用所有的训练数据和实际的测试数据重新运行模型。

```
fm.final <- h2o.deeplearning(
    x = xnames,
    training_frame = h2odigits.train,
    validation_frame = h2odigits.test,
    activation = "Tanh",
    autoencoder = TRUE,
    hidden = hyperparams[[4]]$hidden,
    epochs = 30,
    sparsity_beta = 0,
    input_dropout_ratio = hyperparams[[4]]$input_dr,
    hidden_dropout_ratios = hyperparams[[4]]$hidden_dr,
    l1 = 0,
    l2 = 0
  )

fm.final
Training Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.005880221

H2OAutoEncoderMetrics: deeplearning
** Reported on validation data. **

Validation Set Metrics:
=====================
```

```
MSE: (Extract with 'h2o.mse') 0.006072476
```

我们可以看到，那些根本没有用在训练中的测试数据的 MSE 和训练数据中的 MSE 是相当接近的，尽管稍微有些差，而且在这个情况下，实际上也稍少于来自交叉验证的 MSE 的估计。在某种程度上说，我们搜索到了一个合理的超参数集，这个模型现在是优化了的、验证过的并做好了使用的准备。

在实践中，在使用不同模型或不同超参数集获得更好性能的可能性与运行和训练许多不同模型所用的时间之间，为了平衡这种取舍，通常是很困难的。有时候数据非常大，为了加速计算，使用所有数据的一个随机子集来探索最优模型是很有帮助的。对于本书来说，比起那些常用于深度学习的、通常有着数百万个或数亿个样例的数据集，我们所用的这个案例数据集是相当小的。然而，用在这里的方法可以放大到更大的数据集上，只是会多花上一些时间。值得注意的是，尽管对这些相对小的数据集，我们用相当简单的模型看到了好的性能，更大的数据集可以从复杂模型中获益更多，而且提供了足够的数据去支持学习更复杂的结构。

4.6　小结

本章介绍了有监督学习和无监督学习的区别，包括如何使用无监督学习（比如自动编码器）来学习数据的深度的或隐藏的特征。这些隐藏的特征可用于它们自身，比如更好地理解数据的结构，或者为了其他的应用。自动编码器和无监督学习的两种常见应用是去识别异常数据（例如离群点检测、金融欺诈）以及去预先训练更复杂的、通常是有监督的模型，比如深度神经网络。在第 5 章中，为了开发预测模型（就是说，有监督地学习），我们将学习如何训练并建立深度神经网络。

第 5 章 训练深度预测模型

在本章中，我们将探索如何训练并建立深度预测模型。我们将关注前馈神经网络，它可能是最常见的一个神经网络类型，是一个良好的起点。

本章将涉及下列主题。

- 深度前馈神经网络的入门
- 常见的激活函数：整流器、双曲正切和 maxout
- 选取超参数
- 从深度神经网络训练并预测新数据
- 用例——训练一个深度神经网络用于自动分类

在本章中，我们不会使用任何新的 R 包。唯一的要求是获取 `checkpoint.R` 文件，为剩余要显示的代码建立 R 环境并初始化 H2O 集群。这些都可以用下面的代码来完成。

```
source("checkpoint.R")
options(width = 70, digits = 2)
```

```
cl <- h2o.init(
  max_mem_size = "12G",
  nthreads = 4)
```

5.1 深度前馈神经网络入门

深度前馈神经网络的设计是去近似一个函数 $f()$，这个函数把一些输入变量 x 的集合映射到一个输出变量 y。我们称它为深度前馈神经网络是因为来自输入的信息流要穿过直到输出为止的每一个相继的层，而没有任何反馈和递归循环（既包括前向也包括后向连结的模型被称为循环神经网络（recurrent neural networks））。

深度前馈神经网络适用非常广泛，而且对于诸如图像分类这样的应用特别有用。一般来说，在有一个明确定义的输出时（图像中包含什么数字，是不是有人在上楼梯、下楼梯还是在平地行走，患病/不患病，等等），前馈神经网络对于预测和分类是很有用的。在这些情况中，没有特别的对于反馈循环的需求。循环神经网络对那些反馈循环很重要的情况是有用的，比如说自然语言处理。但是这超过了本书的范围，本书关注的是训练标准的预测模型。

我们可以通过把各层或各函数链接在一起来构建深度前馈神经网络。例如，如图 5-1 中有四个隐藏层的网络。

每个前后相继的层学习一个不同的函数，最后将隐藏层映射到输出。如果一层中包括了足够多的隐藏神经元，它可以将许多不同类型的函数近似到理想的精确程度。即使从最后的隐藏层到输出的映射是带有学习到权重的线性映射，通过先运用从输入层到隐藏层的非线性变换，前馈神经网络也能近似非线性函数。这是深度学习的一种关键力量。例如，在线性回归中，模型学习了从输入到输出的权重。但是这要求必须指定函数形式。在深度前馈神经网络中，从输入层到隐藏层的变换和从隐藏层到输出的变换是同时学习的。本质上，模型学习的是函数形式以及权重。在实践中，虽然模型不可能学习真正的生成模型（generative model），但是它能（非常接近的）近似真正的模型。隐藏神经元越多，近似就越接近。因此对于实践来说，

如果没有理论上的准确性，那它的目标就是模型学习函数的形式。

图 5-1 以有向无环图的形式展示了模型的示意图。作为一种函数表示，来自输入 X 到输出 Y 的全部映射是一个多层函数。第一个隐藏层是 $H_1=f^{(1)}(X,w_1,a_1)$，第二个隐藏层是 $H_2=f^{(2)}(H_1,w_2,a_2)$，以此类推。这些层允许复杂的函数和变换从相对简单的形式开始建立。

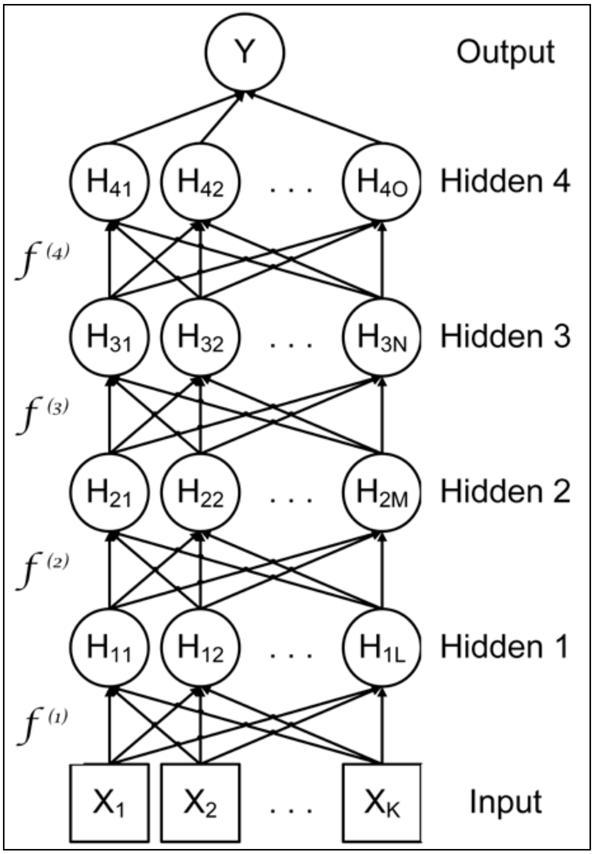


图 5-1

每一层的权重要通过机器来学习，但也依赖于所做的决策，比如说在每一层中应该有多少个隐藏神经元以及应该用什么样的激活函数，这是要在 5.2 节中深入探讨的主题。模型必须要确定的另一个关键部分是成本或损失函数。两个最常用的成

本函数分别是交叉熵（cross-entropy）和二次的函数均方误差（MSE）。

5.2 常用的激活函数——整流器、双曲正切和 maxout

激活函数决定了输入层和隐藏层之间的映射。对于怎样激活神经元，它定义了函数形式。例如，一个线性的激活函数可以定义为 $f(x)=x$，此时神经元的值就是原始输入 x 乘以学习的权重，这是一个线性函数。线性激活函数如图 5-2 上侧的图形所示。确定线性激活函数的问题是，这种激活函数不允许学习任何非线性函数形式。之前，我们曾经使用过双曲正切 $f(x)=\tanh(x)$作为激活函数。双曲正切在某些情况下表现很好，但它有一种潜在的局限，就是在很小或者很大的取值上它就会饱和，这种情况如图 5-2 中间的图形所示。

可能目前最流行的激活函数被称为整流器，可以作为优良的首选（Nair, V., and Hinton, G. E. (2010)）。目前有不同的整流器类型，但最常用的是线性整流器，由函数 $f(x)=\max(0,x)$定义。在某个阈值之下，线性整流器是平坦的，然后变成线性函数，一个例子显示如图 5-2 底部的图形所示。尽管它们形式简单，线性整流器还是提供了一个非线性变换，而且足够的线性整流器能用来近似任意的非线性函数，这和只使用线性激活函数的情况不同。

最后一类要讨论的激活函数是 maxout（Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013)）。maxout 单元取它输入中的最大值，因为通常要在加权之后，所以并不是有最高值的输入变量总会赢。maxout 激活函数再结合丢弃似乎表现特别好。

出于本章的写作目的，我们将重点关注线性整流器。这既是因为它是一种好的、表现优异的缺省值，也是因为在前面的章节中我们已经展示过了双曲正切函数的使用。

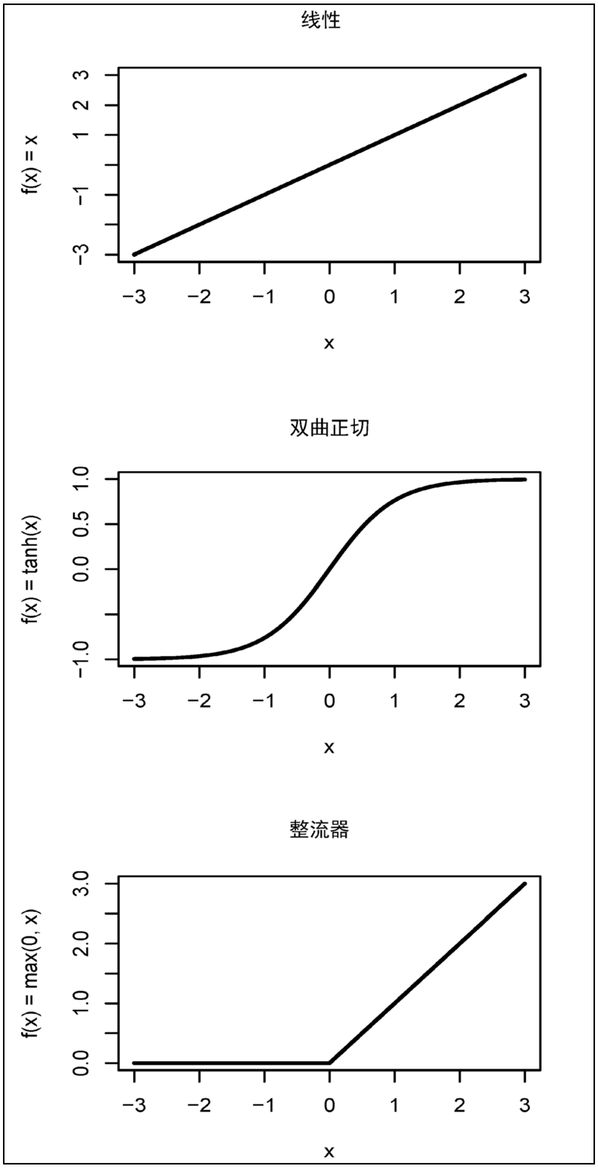

线性
f(x) = x
−3
−1
1
2
3
−3
−2
−1
0
1
2
3
x
双曲正切
f(x) = tanh(x)
−1.0
0.0
0.5
1.0
−3
−2
−1
0
1
2
3
x
整流器
f(x) = max(0, x)
0.0
1.0
2.0
3.0
−3
−2
−1
0
1
2
3
x

图 5-2

5.3 选取超参数

一个模型的参数通常是指诸如权重或者偏置/截距参数这样的内容。但是，还有许多其他的参数必须被设置在偏移上而且没有在模型的训练过程中被优化或被学到。这些参数有时被称为超参数。事实上，甚至模型的选择（例如深度前馈神经网络、随机森林或者支持向量机）也能被视为一种超参数。

即使我们已经以某种方式确定了深度前馈神经网络是最佳的建模策略，依然还有很多超参数需要设置。这些超参数必须明确地要用户指定或者使用由软件所提供的缺省值隐含地指定。

为超参数准备的值在准确度和模型的训练速度上可以有戏剧性的效果。事实上，我们已经见过尝试不同超参数的例子，比如尝试一层中隐藏神经元的个数或者不同的层数。但是，其他超参数也会影响性能和速度。例如，在下列代码中，我们建立 R 环境，载入我们曾经处理过的 MNIST 数据（手写数字的图像）并且运行两个预测模型，只是变化学习率。

```
source("checkpoint.R")
options(width = 70, digits = 2)

cl <- h2o.init(
  max_mem_size = "12G",
  nthreads = 4)

## data setup
digits.train <- read.csv("train.csv")
digits.train$label <- factor(digits.train$label, levels = 0:9)

h2odigits <- as.h2o(
  digits.train,
  destination_frame = "h2odigits")
```

```
i <- 1:32000
h2odigits.train <- h2odigits[i, ]
itest <- 32001:42000
h2odigits.test <- h2odigits[itest, ]
xnames <- colnames(h2odigits.train)[-1]


system.time(ex1 <- h2o.deeplearning(
  x = xnames,
  y = "label",
  training_frame= h2odigits.train,
  validation_frame = h2odigits.test,
  activation = "RectifierWithDropout",
  hidden = c(100),
  epochs = 10,
  adaptive_rate = FALSE,
  rate = .001,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(.2)
))


system.time(ex2 <- h2o.deeplearning(
  x = xnames,
  y = "label",
  training_frame= h2odigits.train,
  validation_frame = h2odigits.test,
  activation = "RectifierWithDropout",
  hidden = c(100),
  epochs = 10,
  adaptive_rate = FALSE,
  rate = .01,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(.2)
))
```

第一个差异是，ex1 相比 ex2 在训练时间上花了 1.34 倍。打印每个模型也显示出了相当大的性能差异。书中为了节约空间，我们忽略了大部分来自 ex1 和 ex2 的输出，只把测试集的度量显示出来。

```
ex1

Test Set Metrics:
=====================
Metrics reported on full validation frame

MSE: (Extract with 'h2o.mse') 0.03326067
R^2: (Extract with 'h2o.r2') 0.9960457
Logloss: (Extract with 'h2o.logloss') 0.2021435
Confusion Matrix: Extract with 'h2o.confusionMatrix(<model>, <
data>)')

=============================================================
         X0   X1  X2   X3  X4  X5   X6   X7  X8  X9      Error
0       984    0   1    0   0   3   13    2   6   2 0.02670623
1         0 1119   5    2   1   1    1    5   5   0 0.01755926
2         7    1 920    8   5   0    6    7   7   2 0.04465213
3         3    5   5 1006   1  13    1    7   7   1 0.04099142
4         0    7   3    0 896   2    5    2   4  13 0.03862661
5         6    2   4   17   5 835    7    1  10   5 0.06390135
6         5    2   1    0   6   8  966    1   2   0 0.02522704
7         2    2   8    7   3   1    0 1027   0   8 0.02930057
8         1   11   3    7   4  15    1    2 922   3 0.04850361
9         5    3   1    7  18   6    2   20   2 932 0.06425703
Totals 1013 1152 951 1054 939 884 1002 1074 965 966 0.03930000

ex2

Test Set Metrics:
=====================
Metrics reported on full validation frame
```

```
MSE: (Extract with 'h2o.mse') 0.1264212
R^2: (Extract with 'h2o.r2') 0.9849702
Logloss: (Extract with 'h2o.logloss') 2.136629
Confusion Matrix: Extract with 'h2o.confusionMatrix(<model>, <
data>)')
==============================================================
         X0   X1  X2   X3  X4   X5   X6   X7   X8  X9      Error
0       938    0   5   11   3   19   19    7    8   1  0.07220574
1         0 1105   6    6   2    6    1    8    5   0  0.02985075
2        18    7 757   54  20    9   47   36    5  10  0.21391485
3         1    2  22  887  10   36    0   50   30  11  0.15443279
4         1    7   0    1 854    7   13    8    5  36  0.08369099
5        11    6   4   45  16  767    8    5   29   1  0.14013453
6        13    5   5    1   6   63  887    5    6   0  0.10494450
7         2    8   3    3   4    7    0 1024    0   7  0.03213611
8         7   48  37   27   8   67   12   22  715  26  0.26212590
9         7    3   3   12  47   22    1  158   11 732  0.26506024
Totals  998 1191 842 1047 970 1003  988 1323  814 824  0.13340000
```

尽管花了更长的时间训练 ex1，在测试数据上它的性能实质上比 ex2 更好。更高的学习率是要更快一些但是也牺牲了性能。这里强调一个需要做出的决策。但是因为这里有许多超参数，所以关于一个超参数的决策不是孤立于其余超参数的。针对这一点的一个例子是正则化。通常，相对更大或者更复杂的模型结合了许多隐藏神经元和可能的多个层来用，选择那些往往会提高准确度（至少在训练数据中）并降低速度的模型。然而，这些复杂的模型通常包括某种形式的正则化，比如说丢弃，往往会降低准确性，因为只有一个子集被包括到任意给定的迭代中。

一个最重要的决定与模型的架构有关。例如，关于应该有多少层，在每一层中应该有多少个隐藏神经元，是否应该有跳跃模式（skipping patterns）或者每一层是否只有顺序连接，等等，必须要做出这样的决策。但是这里没有可以遵循的简单规则来确定这样的问题。

在缺乏学科领域的专长或者先验模型的情况下，设计一个有效的架构要求一些试验和误差。这种试验和误差可以是手动也可以是自动的过程。在理论上，与参数被优化一样，超参数也能被优化。然而，在实践中在计算上这或许是不可行的，因为这要求运行模型太多的变化形式，其中每一个形式都要求有足够的计算时间和资源来完成。

理解每个超参数是什么样有助于启发我们的决定。例如，如果我们从一个模型开始，它的性能比可接受的超参数更差，应该改变成在模型中允许更大的容量和灵活性，例如增加更多的隐藏神经元、隐藏神经元额外的层、更多的训练轮数等。如果在训练数据和测试数据上的模型性能有较大差异，或许表明模型过拟合了数据，这种情况下我们可以调整超参数，降低容量或者加上更多的正则化。在某些情况下，我们可以要求更多的数据去支持拟合一个需要适当预测结果的更复杂的模型。在第 6 章中，我们将讨论一些改善模型架构的方法（包括更分析化的方法）的更多细节。

5.4 从深度神经网络训练和预测新数据

在本节，我们将学习如何训练深度神经网络并使用它们来生成新数据上的预测。用于本节的例子将使用我们曾经处理过的活动数据，下列代码简单地建立了数据。

```
use.train.x <- read.table("UCI HAR Dataset/train/X_train.txt")
use.test.x <- read.table("UCI HAR Dataset/test/X_test.txt")

use.train.y <- read.table("UCI HAR Dataset/train/y_train.txt")
[[1]]
use.test.y <- read.table("UCI HAR Dataset/test/y_test.txt")[[1
]]

use.train <- cbind(use.train.x, Outcome = factor(use.train.y))
use.test <- cbind(use.test.x, Outcome = factor(use.test.y))
```

```
use.labels <- read.table("UCI HAR Dataset/activity_labels.txt")

h2oactivity.train <- as.h2o(
  use.train,
  destination_frame = "h2oactivitytrain")

h2oactivity.test <- as.h2o(
  use.test,
  destination_frame = "h2oactivitytest")
```

我们已经学习了训练一个深度神经网络的各个组成部分。就像对自动编码器的处理一样，我们使用 `h2o.deeplearning()` 函数，但是对参数 x 和 y 都指定变量名。之前，我们引入了测试数据来自动地在训练和测试数据上得到性能度量。但是，为了显示在新数据上如何生成预测，我们不把它包括在 `h2o.deeplearning()` 的调用中。这里使用的激活函数是线性整流器，带有在输入变量上的（20%）和隐藏神经元上的（50%）丢弃。这个小例子是只有 50 个隐藏神经元和 10 次训练迭代的一个浅层网络。成本（损失）函数是交叉熵。

```
mt1 <- h2o.deeplearning(
  x = colnames(use.train.x),
  y = "Outcome",
  training_frame= h2oactivity.train,
  activation = "RectifierWithDropout",
  hidden = c(50),
  epochs = 10,
  loss = "CrossEntropy",
  input_dropout_ratio = .2,
  hidden_dropout_ratios = c(.5), ,
  export_weights_and_biases = TRUE
)
```

在 R 操作台简单地键入名称，我们看到了存储的对象。第一个信息是关于模型

类型的。结果有 6 个离散的水平，所以要用到一个多项式模型。模型一共包括 28 406 个权重/偏差（biase）。偏差就像截距或者常数的偏移。因为这是一个前馈神经网络，只在相邻的层之间有权重。输入变量没有偏差，但隐藏神经元和输出有。28 406 个权重的构成中有来自于输入变量和第一层隐藏神经元之间的$561 * 50 = 28 050$个权重，来自于隐藏神经元和输出之间的$50 * 6 = 300$个权重（6 是因为结果中有六个不同的水平）以及来自于隐藏神经元的 50 个偏差和来自输出的 6 个偏差。

输出也显示了层数和每一层中单元的个数、每一个单元的类型、丢弃的百分比以及其他的正则和超参数信息。

```
mt1
Model Details:
==============

H2OMultinomialModel: deeplearning
Model ID: DeepLearning_model_R_1451894068318_16
Status of Neuron Layers: predicting Outcome, 6-class classification,
multinomial distribution, CrossEntropy loss, 28,406 weights/
biases, 406.9
KB, 73,520 training samples, mini-batch size 1
  layer units           type dropout       l1       l2 mean_rate
1   1   561            Input 20.00 %
2   2    50 RectifierDropout 50.00 % 0.000000 0.000000  0.001891
3   3     6          Softmax         0.000000 0.000000  0.004912
  rate_RMS momentum mean_weight weight_RMS mean_bias bias_RMS
1
2 0.002408 0.000000    0.000172   0.062088   0.347545  0.114483
3 0.015856 0.000000   -0.009241   0.755695  -0.029887  0.294392
```

接下来的一组输出报告了训练数据上的性能度量，包括均方误差（越低越好）、R^2（越高越好）以及对数损失（越低越好）。

```
H2OMultinomialMetrics: deeplearning
** Reported on training data. **
```

```
    Description: Metrics reported on temporary (load-balanced)
training frame

    Training Set Metrics:
    =====================
    Metrics reported on temporary (load-balanced) training frame

    MSE: (Extract with 'h2o.mse') 0.023
    R^2: (Extract with 'h2o.r2') 0.99
    Logloss: (Extract with 'h2o.logloss') 0.082
```

最后，代码打印出来一个混淆矩阵，显示了真实输出对比预测输出的情况。行中显示的是观测的输出，列中显示的是预测的输出。对角线标示了正确的分类并根据结果水平显示了误差率。

```
    Confusion Matrix: Extract with 'h2o.confusionMatrix(<model>,
train =TRUE)')
    ==============================================================

              X1    X2    X3   X4    X5    X6  Error         Rate
    1       1216    10     0    0     0     0  0.0082   10 / 1,226
    2          3  1070     0    0     0     0  0.0028    3 / 1,073
    3          2    11   973    0     0     0  0.0132     13 / 986
    4          0     1     0 1236    40     9  0.0389   50 / 1,286
    5          0     0     0  146  1228     0  0.1063  146 / 1,374
    6          0     0     0    0     0  1407  0.0000    0 / 1,407
    Totals  1221  1092   973 1382  1268  1416  0.0302  222 / 7,352

    Hit Ratio Table: Extract with 'h2o.hit_ratio_table(<model>,train
 = TRUE)'
    ==============================================================
      Top-6 Hit Ratios:
        k   hit_ratio
    1   1    0.969804
    2   2    0.999728
```

```
3    3     1.000000
4    4     1.000000
5    5     1.000000
6    6     1.000000
```

我们能使用 h2o.deepfeatures() 函数提取并查看模型的特征，指定模型、数据以及我们想要提取的层数。下列代码提取了特征并查看了开头几行。默认情况下结果也包括在内。注意特征中的零，它们在结果中是因为我们使用了线性整流器，低于零的值被删为零。

```
f <- as.data.frame(h2o.deepfeatures(mt1, h2oactivity.train, 1))
f[1:10, 1:5]
  Outcome  DF.L1.C1  DF.L1.C2  DF.L1.C3  DF.L1.C4
1       5      0.00       5.9     0.091       2.1
2       5      0.00       4.7     0.000       1.7
3       5      0.00       4.4     0.102       1.5
4       5      0.00       4.9     0.000       1.9
5       5      0.00       5.0     0.000       1.8
6       5      0.00       4.9     0.000       2.0
7       5      0.00       4.9     0.000       1.6
8       5      0.00       4.6     0.000       1.8
9       5      0.00       5.0     0.000       1.6
10      5      0.13       5.1     0.000       1.3
```

就像提取特征一样，我们也能提取每一层的权重。下列代码提取了权重并做出一张热图，这样我们就能看到这里是否存在确定的输入变量对特别的神经元有着更高权重的清晰模式。

```
w1 <- as.matrix(h2o.weights(mt1, 1))

## plot heatmap of the weights
tmp <- as.data.frame(t(w1))
tmp$Row <- 1:nrow(tmp)
tmp <- melt(tmp, id.vars = c("Row"))
```

```
p.heat <- ggplot(tmp,
       aes(variable, Row, fill = value)) +
  geom_tile() +
  scale_fill_gradientn(colours = c("black", "white", "blue")) +
  theme_classic() +
  theme(axis.text = element_blank()) +
  xlab("Hidden Neuron") +
  ylab("Input Variable") +
  ggtitle("Heatmap of Weights for Layer 1")
print(p.heat)
```

由图 5-3 可见，对于特定神经元主要由少数输入构成的效应，似乎没有特别清楚的模式。

对于所有的复杂性来说，一旦模型被训练为前馈神经网络，就直接评分并用于生成数据上的预测。这里有内置函数来完成这一点，但为了更好地理解模型，我们将手动处理一个例子。

就像之前所注意的那样，前馈神经网络是由各个 `layering` 函数一起构建的。我们已经对第一层提取了权重。但是，为了构建第一个隐藏层的神经元，我们还需要输入数据和偏差。因为需要在一整列当中加入一些常数项来构建深度特征（即使是偏差也要存储在一个向量中，每个隐藏神经元有一个偏差），我们复制偏差并把它转化到一个维数与输入数据相匹配的矩阵中。

```
## input data
d <- as.matrix(use.train[, -562])

## biases for hidden layer 1 neurons
b1 <- as.matrix(h2o.biases(mt1, 1))
b12 <- do.call(rbind, rep(list(t(b1)), nrow(d)))
```

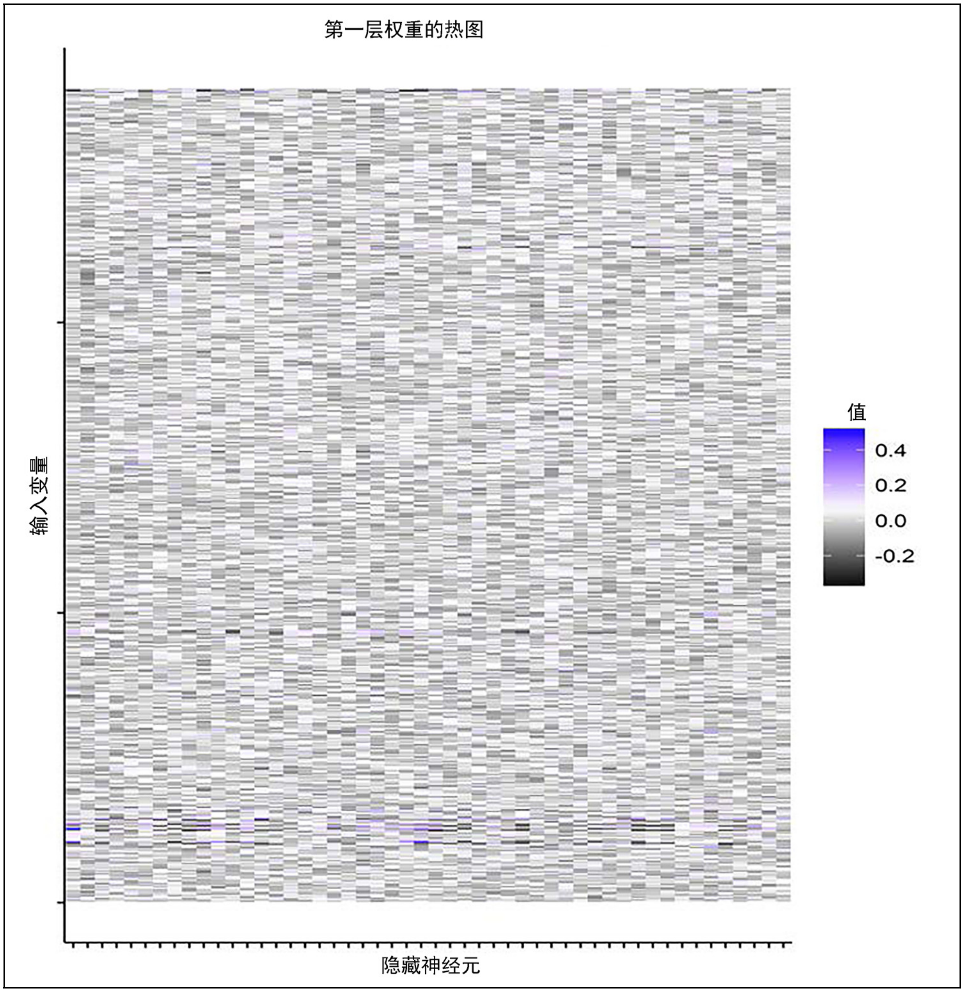


图 5-3

现在我们能对第一层的隐藏神经元构建特征。首先，我们需要把输入数据的每一列标准化，在 R 中对数据逐列（矩阵的第二个维度）运用 scale() 函数来实现这一点。

```
d.scaled <- apply(d, 2, scale)
```

接下来，我们用之前提取的权重右乘以标准化的数据，然后再加上偏差矩阵。

```
d.weighted <- d.scaled %*% t(w1) + b12
```

因为在隐藏层上引入了丢弃，我们需要运用一种修正。这仅仅是一种基于隐藏单元比例的乘法修正，可以被包括到任何迭代中——即，1-丢弃比例。

```
d.weighted <- d.weighted * (1 - .5)
```

最后，因为我们使用的是线性整流器，对每一列，我们只想取零或高于零的值。在 R 中对加权数据逐列运用 `pmax()` 函数，我们实现了这个结果。

```
d.weighted.rectifier <- apply(d.weighted, 2, pmax, 0)
```

我们把处理结果和由 H2O 提取的特征进行比较，能够检查它是否正确。我们使用 `all.equal()` 函数进行比较，对由浮点运算所引起的轻微数值差异给予一些容忍度。

```
all.equal(
  as.numeric(f[, 2]),
  d.weighted.rectifier[, 1],
  check.attributes = FALSE,
  use.names = FALSE,
  tolerance = 1e-04)
```

用相似的方式，我们能对下一层即输出层提取权重和偏差。就像创建隐藏神经元一样，乘以权重并加上偏差，我们创建了预测的结果。然而，这些操作并不是运用到原始数据上的，而是运用到我们在第一层所构建的特征上的。和以前一样，我们需要对恰当的维度扩展偏差。

```
w2 <- as.matrix(h2o.weights(mt1, 2))

b2 <- as.matrix(h2o.biases(mt1, 2))
b22 <- do.call(rbind, rep(list(t(b2)), nrow(d)))

yhat <- d.weighted.rectifier %*% t(w2) + b22
```

为了构建隐藏神经元，我们使用线性整流器激活函数。对于输出来说，我们使用了 softmax 函数，它将所有的预测标准化到区间[0,1]当中并确保它们的和等于1，就像是一种预测的概率。我们知道，使用 normalizes 函数一是因为它很常用；二是因为在较早的模型输出中，H2O 表明 softmax 是链接到输出层的函数。softmax 函数是对每一种情况来定义的，而且是指数预测除以这种情况下指数预测的和。

```
yhat <- exp(yhat)
normalizer <- do.call(cbind, rep(list(rowSums(yhat)), ncol
(yhat)))
yhat <- yhat / normalizer
```

最后，通过使用 which.max()函数添加到预测数据集，选择具有最高预测概率的输出列，我们能得到预测的分类。

```
yhat <- cbind(Outcome = apply(yhat, 1, which.max), yhat)
```

凭借 h2o.predict()函数，我们也能提取使用了内置函数的预测，而且能把这些预测和我们手动生成的预测进行比较。

```
yhat.h2o <- as.data.frame(h2o.predict(mt1, newdata = h2oactivity.
train))

xtabs(~ yhat[, 1] + yhat.h2o[, 1])

           yhat.h2o[, 1]
yhat[, 1]      1      2     3      4      5      6
        1   1216      0     0      0      0      0
        2      0   1122     0      0      0      0
        3      0      0   948      0      0      0
        4      0      0     0   1316      0      0
        5      0      0     0      0   1344      0
        6      0      0     0      0      0   1406
```

我们手动地处理完全匹配了 H2O 的结果。当然，在实践中并不需要手动地重新实现预测函数，而且展示手动处理的代码也并不特别具有计算效率。然而，通过处理一个类似这样的例子，有助于帮助我们完全搞清楚什么样的部分进入了模型以及它们是如何使用的。如果我们有许多神经元的隐藏层，处理会非常相似，只要对每一层重复这些步骤来生成特征，而且总是建立在上一层的结果之上。

5.5 用例——为自动分类生成深度神经网络

对于用例，我们使用的数据是百万歌曲数据集（Million Song Dataset）的一个子集，来自于加州大学欧文分校在线数据仓库（University of California Irvine online dataset repository，Lichman, M. (2013)）。数据集中有 515 345 个样例，其中前 463 715 个用作训练样例，后 51 630 个样例用作测试。这个数据集的第一列包含年份，其余各列是来自歌曲音色的特征。我们可以在 http://archive.ics.uci.edu/ml/ datasets/ YearPredictionMSD 下载并解压数据。我们的目的是预测每首歌发行的年份。

```
download.file(
"http://archive.ics.uci.edu/ml/machine-learning-databases/00203/
YearPredictionMSD.txt.zip", destfile = "YearPredictionMSD.txt.
zip")
unzip("YearPredictionMSD.txt.zip")
```

使用来自 data.table 包的 `fread()` 函数，我们能将数据读入到 R 中。在这里，`fread()` 函数比 `read.csv()` 函数略胜一筹，因为它要快三个数量级，在带有固态硬盘的高端台式电脑中只需要耗费 30 秒。

```
d <- fread("YearPredictionMSD.txt", sep = ",")
```

首先，我们快速看一下结果——发行年份的分布。下面的代码创建了一个直方图，如图 5-4 所示。

```
p.hist <- ggplot(d[, .(V1)], aes(V1)) +
  geom_histogram(binwidth = 1) +
  theme_classic() +
  xlab("Year of Release")
print(p.hist)
```

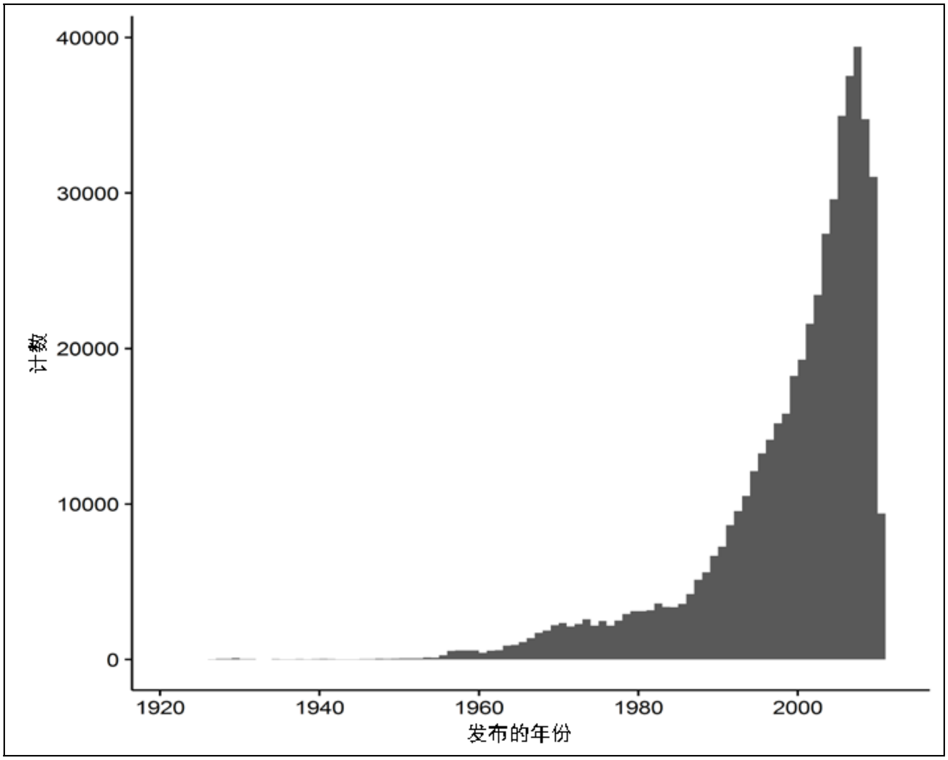


图 5-4

一个可能需要关注的地方是，相对极端的值可能对模型产生不恰当的影响。通过考虑数据分布并取平方根，我们能减少这种影响。我们还能排除掉少量更极端的观测，比如排除最小和最大的各 0.5%的观测（合计 1%）。我们（用下列代码）检查可能包括 1957 年到 2010 年之间年份的分位数。

```
quantile(d$V1, probs = c(.005, .995))
0.5% 100%
1957 2010
```

下列代码整理了数据并为 H2O 转化了训练和测试数据集。

```
d.train <- d[1:463715][V1 >= 1957 & V1 <= 2010]

d.test <- d[463716:515345][V1 >= 1957 & V1 <= 2010]

h2omsd.train <- as.h2o(
  d.train,
  destination_frame = "h2omsdtrain")

h2omsd.test <- as.h2o(
  d.test,
  destination_frame = "h2omsdtest")
```

我们可以建立一个线性回归模型，作为起点并提供某种基准的性能水平。

```
summary(m0 <- lm(V1 ~ ., data = d.train))$r.squared
[1] 0.24

cor(
  d.test$V1,
  predict(m0, newdata = d.test))^2
[1] 0.23
```

尽管不是很大，线性回归在训练数据上考虑了年份上 24%的变化，而在测试数据上则是 23%。这个结果为我们提供了一个和前馈神经网络进行对照的基准。

我们的第一个网络是只有一个隐藏层的浅层网络，它是相当小的。这个数据集比我们之前处理过的一些数据集更大，但依然是足够小的，可以轻松地对它进行全部处理。为了做出在全部数据集上发生的性能评分，我们使用一个特别的值，0，来传递给 `score_training_samples` 和 `core_validation_samples` 参数。在 10 核 H2O 集群的设备上，模型训练用了 79 秒，使用 `system.time()`函数记录时间。

```
m1 <- h2o.deeplearning(
  x = colnames(d)[-1],
  y = "V1",
  training_frame= h2omsd.train,
  validation_frame = h2omsd.test,
  activation = "RectifierWithDropout",
  hidden = c(50),
  epochs = 100,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(0),
  score_training_samples = 0,
  score_validation_samples = 0,
  diagnostics = TRUE,
  export_weights_and_biases = TRUE,
  variable_importances = TRUE
  )
```

来自于这个简单模型的结果显示了它比线性回归有明显的性能提升。前馈神经网络即使只有一个带有 50 个隐藏神经元的层，在测试数据中考虑了发布年份的 32% 的变化，高于使用线性回归的 23%。

因为这个模型是小的，而且隐藏神经元比输入变量还少，这里没有使用丢弃或者其他的正则化。然而，训练数据和测试数据之间的性能上的一些矛盾之处（R^2=0.37 对 R^2=0.32），表明某种正则化是有帮助的。

```
m1

Model Details:
==============
H2ORegressionModel: deeplearning
Model ID:  DeepLearning_model_R_1451972322936_5
Status of Neuron Layers: predicting V1, regression, gaussian
distribution, Quadratic loss, 4,601 weights/biases, 72.5 KB,
13,702,476
```

```
training samples, mini-batch size 1
  layer units              type  dropout       l1       l2  mean_rate
1     1    90             Input   0.00 %
2     2    50 RectifierDropout    0.00 % 0.000000 0.000000  0.009403
3     3     1            Linear          0.000000 0.000000  0.000218
  rate_RMS momentum mean_weight weight_RMS mean_bias bias_RMS
1
2 0.007939 0.000000   -0.018219   0.598229 -2.199141 2.245173
3 0.000202 0.000000   -0.042807   0.103305 -0.767868 0.000000


H2ORegressionMetrics: deeplearning
** Reported on training data. **
Description: Metrics reported on full training frame

MSE: 76
R2 : 0.37
Mean Residual Deviance : 76


H2ORegressionMetrics: deeplearning
** Reported on validation data. **
Description: Metrics reported on temporary (load-balanced)
validation frame

MSE: 80
R2 : 0.32
Mean Residual Deviance : 80
```

虽然浅层神经网络比线性回归性能有所提升，但它的表现依然不太好，性能还有明显提升的空间。接下来，我们将尝试一个更大的深度前馈神经网络。在接下来的模型代码中，我们有三个隐藏神经元的层，分别有 200 个、200 个、400 个隐藏神经元。我们将在隐藏层（而不是输入层）上引入一个中等数量的丢弃。这个模型的训练耗费了 843 秒。

```
m2 <- h2o.deeplearning(
  x = colnames(d)[-1],
  y = "V1",
  training_frame= h2omsd.train,
  validation_frame = h2omsd.test,
  activation = "RectifierWithDropout",
  hidden = c(200, 200, 400),
  epochs = 100,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(.2, .2, .2),
  score_training_samples = 0,
  score_validation_samples = 0,
  diagnostics = TRUE,
  export_weights_and_biases = TRUE,
  variable_importances = TRUE
  )
```

检查这个模型的性能，比起我们开始尝试的小的浅层网络，有了显而易见的提升。在测试数据中，浅层网络有 0.32 的 R^2，而在深度网络中有 0.35 的 R^2。

这里依然有某种程度的过拟合。这个模型在训练数据和测试数据中 R^2 的差异是 0.05，我们可以与这个差异同样是 0.05 的更简单的模型比较。更复杂的模型提高了性能，在过拟合上相差无几，这可能要归功于丢弃的使用。

```
m2

Model Details:
==============

H2ORegressionModel: deeplearning
Model ID:  DeepLearning_model_R_1452031055473_5
Status of Neuron Layers: predicting V1, regression, gaussian
distribution, Quadratic loss, 139,201 weights/biases, 1.6 MB,
22,695,351
training samples, mini-batch size 1
```

```
  layer units              type  dropout        l1        l2  mean_rate
1   1    90             Input   0.00 %
2   2   200 RectifierDropout  20.00 % 0.000000 0.000000 0.011513
3   3   200 RectifierDropout  20.00 % 0.000000 0.000000 0.014861
4   4   400 RectifierDropout  20.00 % 0.000000 0.000000 0.054338
5   5     1            Linear          0.000000 0.000000 0.001258
  rate_RMS momentum mean_weight weight_RMS mean_bias bias_RMS
1
2 0.004978 0.000000    0.000848   0.207373 -0.254659 0.321144
3 0.012359 0.000000   -0.032566   0.104347  1.017329 0.341556
4 0.036596 0.000000   -0.031768   0.072171  0.651546 0.292565
5 0.000505 0.000000    0.001421   0.020867 -0.596303 0.000000


H2ORegressionMetrics: deeplearning
** Reported on training data. **
Description: Metrics reported on full training frame

MSE:  66
R2 :  0.40
Mean Residual Deviance :  66


H2ORegressionMetrics: deeplearning
** Reported on validation data. **
Description: Metrics reported on temporary (load-balanced)
validation frame

MSE:  70
R2 :  0.35
Mean Residual Deviance :  70
```

为了查看性能是否在测试数据上有更多的提高，我们可以尝试一个额外的模型，包括了每一层中实质上更多的隐藏神经元、更多的训练轮数以及带有更高程度

的正则化。大家可能不想运行下列代码（在 10 核 H2O 集群上要花费多于 10 小时的时间）。

```
m3 <- h2o.deeplearning(
  x = colnames(d)[-1],
  y = "V1",
  training_frame= h2omsd.train,
  validation_frame = h2omsd.test,
  activation = "RectifierWithDropout",
  hidden = c(500, 500, 1000),
  epochs = 500,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(.5, .5, .5),
  score_training_samples = 0,
  score_validation_samples = 0,
  diagnostics = TRUE,
  export_weights_and_biases = TRUE
  )
```

这个模型在测试数据上的性能事实上比前两个模型要差，尽管它依然优于线性回归。

```
m3

Model Details:
==============

H2ORegressionModel: deeplearning
Model ID:  DeepLearning_model_R_1451972322936_15
Status of Neuron Layers: predicting V1, regression, gaussian
distribution, Quadratic loss, 798,001 weights/biases, 9.2 MB,
47,002,720
training samples, mini-batch size 1
layer units             type dropout       l1       l2 mean_rate
1   1    90            Input  0.00 %
```

```
2   2    500 RectifierDropout 50.00 %  0.000000 0.000000 0.028872
3   3    500 RectifierDropout 50.00 %  0.000000 0.000000 0.047632
4   4   1000 RectifierDropout 50.00 %  0.000000 0.000000 0.084886
5   5      1           Linear          0.000000 0.000000 0.001238
  rate_RMS momentum mean_weight weight_RMS mean_bias bias_RMS
1
2 0.014727 0.000000   0.000941   0.069018  0.417255 0.048082
3 0.020226 0.000000  -0.007515   0.049535  0.968111 0.054521
4 0.062396 0.000000  -0.009451   0.038735  0.929930 0.032726
5 0.000445 0.000000   0.000538   0.014785 -0.478095 0.000000


H2ORegressionMetrics: deeplearning
** Reported on training data. **
Description: Metrics reported on full training frame

MSE:  84
R2 :  0.30
Mean Residual Deviance :  84


H2ORegressionMetrics: deeplearning
** Reported on validation data. **
Description: Metrics reported on temporary (load-balanced)
validation frame

MSE:  85
R2 :  0.28
Mean Residual Deviance :  85
```

最好的模型依然是深度模型，但只是每一层有较少神经元的深度模型。我们能够使用一种方法试着查看模型性能是否被提升，即对额外的轮数或者迭代尝试训练。模型的输入中是模型的 ID。对于最佳表现的模型，是 DeepLearning_model_R_1452031055473_5。它可以传递给h2o.deeplearning()函数的checkpoint

参数，这样训练可以开始使用来自之前模型的权重。注意，我们运行代码的时候，每一次的模型 ID 都会不同。因此，当我们在自己的电脑或服务器上运行代码的时候，需要使用来自于我们所运行的模型的 ID。

既然一般的架构——隐藏神经元个数、层数和连结——保持相同，使用检查点可以极大地节约时间。这既是因为之前的训练迭代可以被重复使用，也是因为通常较早的迭代会比较晚的迭代花费更长的时间。下面的例子展示了如何运行模型将轮数从 500 改变到 1 000（因为 500 已经实现了），而且将模型名字指定为字符串传递给 checkpoint 参数，从前一个模型开始运行。

```
m2b <- h2o.deeplearning(
  x = colnames(d)[-1],
  y = "V1",
  training_frame= h2omsd.train,
  validation_frame = h2omsd.test,
  activation = "RectifierWithDropout",
  hidden = c(200, 200, 400),
  checkpoint = "DeepLearning_model_R_1452031055473_5",
  epochs = 1000,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(.2, .2, .2),
  score_training_samples = 0,
  score_validation_samples = 0,
  diagnostics = TRUE,
  export_weights_and_biases = TRUE,
  variable_importances = TRUE

  )
```

然而最终，额外的轮数并没有提升模型性能。事实上，它变得更差了。

```
m2b

Model Details:
```

```
==============

H2ORegressionModel: deeplearning
Model ID:  DeepLearning_model_R_1452031055473_81

Status of Neuron Layers: predicting V1, regression, gaussian
distribution, Quadratic loss, 139,201 weights/biases, 1.6 MB,
30,054,531
training samples, mini-batch size 1
  layer units             type dropout       l1       l2  mean_rate
1     1    90            Input  0.00 %
2     2   200 RectifierDropout 20.00 % 0.000000 0.000000  0.008598
3     3   200 RectifierDropout 20.00 % 0.000000 0.000000  0.012581
4     4   400 RectifierDropout 20.00 % 0.000000 0.000000  0.025138
5     5     1           Linear         0.000000 0.000000  0.000895
  rate_RMS momentum mean_weight weight_RMS mean_bias bias_RMS
1
2 0.004485 0.000000   -0.004116   0.473692 -1.601533 1.060434
3 0.017790 0.000000   -0.040249   0.239924  0.767950 1.305716
4 0.022843 0.000000   -0.048592   0.105753  0.360921 0.439503
5 0.000582 0.000000   -0.001778   0.029287 -0.065273 0.000000


H2ORegressionMetrics: deeplearning
** Reported on training data. **
Description: Metrics reported on full training frame

MSE:  62
R2 :  0.43
Mean Residual Deviance :  62


H2ORegressionMetrics: deeplearning
** Reported on validation data. **
Description: Metrics reported on temporary (load-balanced)
```

```
validation frame

   MSE:  72
   R2 :  0.33
   Mean Residual Deviance :  72
```

将模型保存在 R 中是很容易的，但是从 R 调用 H2O 时，大多数结果并没有真的存贮在 R 当中，相反它们是存贮在 H2O 集群中的。因此，仅仅保存 R 对象只不过是保存了 H2O 中模型的引用，而且如果这种引用关闭或者丢失，完整的模型结果将不被保存下来。为了避免这种情况并保存完整的模型结果，我们使用 `h2o.saveModel()` 函数并指定要保存的模型（通过把它传递给 R 对象）、路径 `path` 以及当文件存在的时候是否要覆盖文件（使用 `force = TRUE`）。

```
h2o.saveModel(
  object = m2,
  path = "c:\\Users\\jwile\\DeepLearning",
  force = TRUE)
```

这将创建一个目录，包括所有需要载入并再次使用模型的文件。一旦我们保存了模型，就可以使用 `h2o.loadModel()` 函数将这个模型加载到一个新的 H2O 集群中。注意，对于要加载的模型结果，我们必须指定文件夹的名称。

除了保存模型结果以便再次载入到 H2O 集群之外，我们还可以将模型保存成一个简单的 Java 对象（Plain Old Java Object，POJO）。把模型保存成 POJO 是有用的，因为它们能内嵌到任何应用中并且使用评分结果。H2O 模型可以使用 `h2o.download_pojo()` 函数，以同样的参数保存成 POJO。

另一个有用的函数是 `h2o.scoreHistory()`。评分历史显示了模型在各次迭代中的性能、时间戳以及每一个轮数的持续时间。下列代码显示了如何使用这个函数和结果。

```
h2o.scoreHistory(m2)
```

```
Scoring History:
            timestamp          duration training_speed   epochs
1  2016-01-06 23:20:18         0.000 sec                 0.00000
2  2016-01-06 23:20:26        15.537 sec 13922 rows/sec  0.21687
3  2016-01-06 23:21:51  1 min 40.761 sec 22603 rows/sec  4.11902
4  2016-01-06 23:23:15  3 min  4.790 sec 25030 rows/sec  8.66890
5  2016-01-06 23:24:39  4 min 28.208 sec 26347 rows/sec 13.43506
6  2016-01-06 23:26:00  5 min 49.401 sec 27540 rows/sec 18.41458
7  2016-01-06 23:27:21  7 min 10.032 sec 28317 rows/sec 23.39553
8  2016-01-06 23:28:40  8 min 29.325 sec 28928 rows/sec 28.37323
9  2016-01-06 23:29:59  9 min 48.908 sec 29354 rows/sec 33.34907
10 2016-01-06 23:31:21 11 min 10.056 sec 29771 rows/sec 38.54472
11 2016-01-06 23:32:41 12 min 30.532 sec 30130 rows/sec 43.73626
12 2016-01-06 23:34:04 13 min 53.652 sec 30444 rows/sec 49.14818
13 2016-01-06 23:34:12 14 min  1.667 sec 30442 rows/sec 49.14818
    iterations        samples training_MSE training_deviance
1            0       0.000000
2            1   100145.000000     73.50950          73.50950
3           19  1902057.000000     65.90201          65.90201
4           40  4003071.000000     66.39865          66.39865
5           62  6203960.000000     63.97995          63.97995
6           85  8503375.000000     65.20361          65.20361
7          108 10803448.000000     62.67372          62.67372
8          131 13102020.000000     63.91678          63.91678
9          154 15399734.000000     60.31355          60.31355
10         178 17798949.000000     60.15803          60.15803
11         202 20196268.000000     61.71012          61.71012
12         227 22695351.000000     58.34747          58.34747
13         227 22695351.000000     65.90201          65.90201
    training_r2 validation_MSE validation_deviance validation_r2
1
2       0.32564       73.67272            73.67272       0.30763
3       0.39543       69.57711            69.57711       0.34612
4       0.39087       71.70615            71.70615       0.32611
5       0.41306       70.45211            70.45211       0.33790
```

```
6        0.40184     71.98921          71.98921      0.32345
7        0.42505     70.90519          70.90519      0.33364
8        0.41364     72.69913          72.69913      0.31678
9        0.44670     70.49905          70.49905      0.33746
10       0.44812     70.76801          70.76801      0.33493
11       0.43389     72.22494          72.22494      0.32124
12       0.46473     70.55234          70.55234      0.33696
13       0.39543     69.57711          69.57711      0.34612
```

到目前为止，我们只检查了模型的整体性能。尽管这是一种有用的概括，但它提供的还不是完整的图景。检查残差有助于我们理解，在数据范围内性能是否具有一致性以及是否有任何异常的方差。这些结果有助于我们更全面的评价性能。我们能计算残差，方法是使用 `h2o.predict()` 函数得到所有观测的预测值，然后取观测值和预测值的差。下面的代码提取了预测值，把它们和观测值合并在一起并且画图。残差为零象征着一个完美的预测，正的或者负的残差象征着预测过度或者预测不足。因为年份是离散的，使用下面的代码，我们能对每首歌发布的真实年份采用残差的箱形图来可视化这个数据，如图 5-5 所示。

```
yhat <- as.data.frame(h2o.predict(m1, h2omsd.train))
yhat <- cbind(as.data.frame(h2omsd.train[["V1"]]), yhat)

p.resid <- ggplot(yhat, aes(factor(V1), predict - V1)) +
  geom_boxplot() +
  geom_hline(yintercept = 0) +
  theme_classic() +
  theme(axis.text.x = element_text(
          angle = 90, vjust = 0.5, hjust = 0)) +
  xlab("Year of Release") +
  ylab("Predicted Year of Release")
print(p.resid)
```

结果表明，在后面的年份中有一种残差减少的显著模式，或者换种说法，它表明了前面的年份有极端异常的模型预测。在某种程度上，这或许要归咎于数据的分

布。对于从 1990 年代中期到 2000 年代的大多数观测，如同之前我们在图 5-4 中所见到的，对于准确地预测这些值，模型是最敏感的，在 1990 年或 1980 年之前较少的观测具有较小的影响。

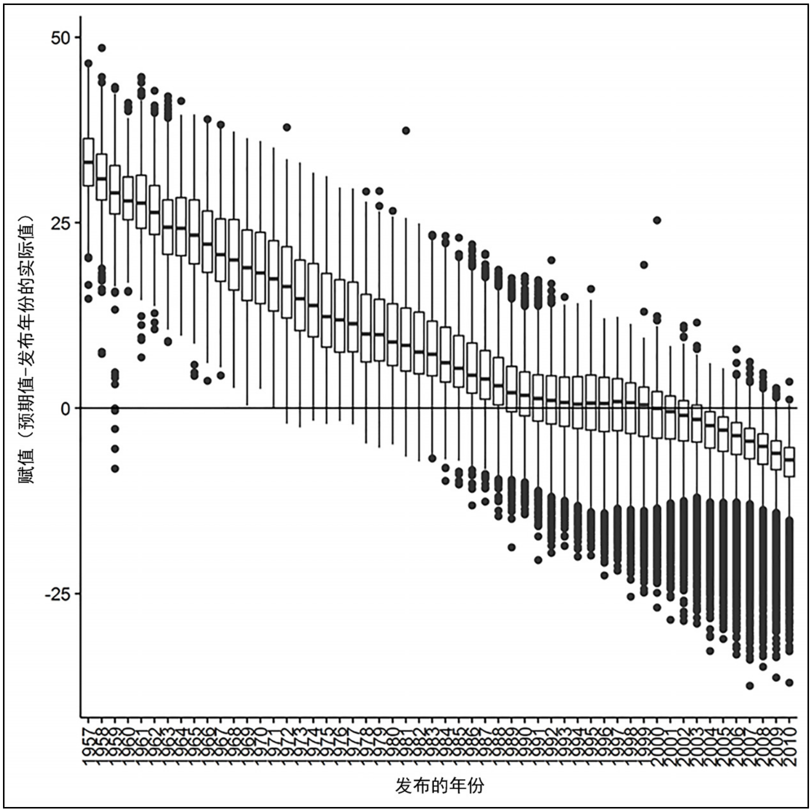


图 5-5

因为使用了 `variable_importances` 参数，我们使用 `h2o.varimp()` 函数能为模型提取每个变量的相对重要性。尽管精确分配每个变量的重要性是很困难的，但它有助于提供一种大致的感觉，判断哪个变量能比其他变量对预测做出更大的贡献。例如，为了排除一些贡献非常小的变量，这可能是一种有益的方式。下面的代码提取了重要的变量，打印了最重要的 10 个（数据集按照从最重要到最不重要的方式排列），而且做了结果的图形来演示分布，如图 5-6 所示。

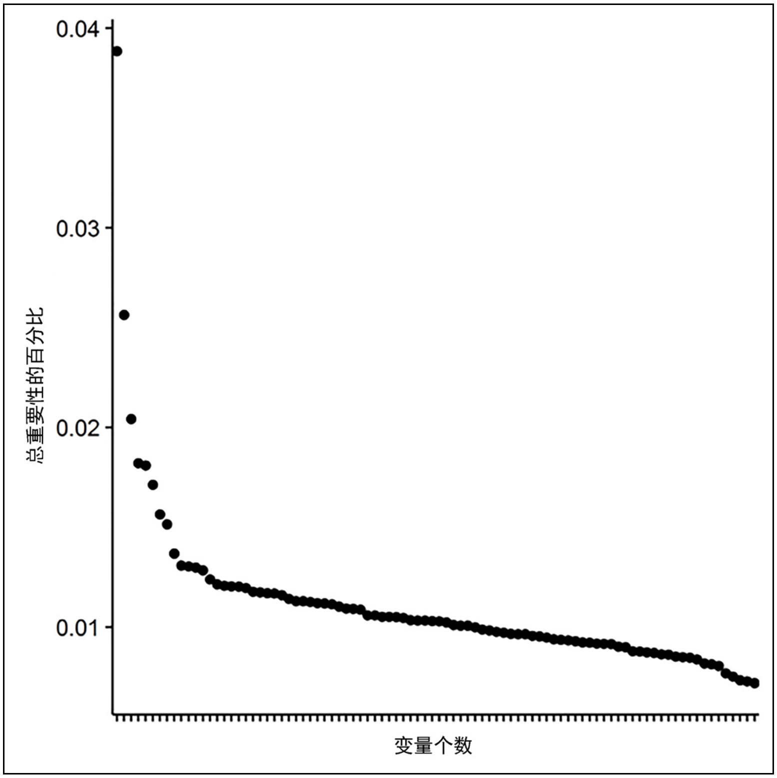


图 5-6

```
imp <- as.data.frame(h2o.varimp(m2))
imp[1:10, ]
    variable relative_importance scaled_importance percentage
1         V2                1.00              1.00      0.039
2         V3                0.66              0.66      0.026
3         V4                0.53              0.53      0.020
4        V14                0.47              0.47      0.018
5        V24                0.47              0.47      0.018
6         V7                0.44              0.44      0.017
7        V37                0.40              0.40      0.016
8         V6                0.39              0.39      0.015
9        V59                0.35              0.35      0.014
10       V26                0.34              0.34      0.013
```

```
p.imp <- ggplot(imp, aes(factor(variable, levels = variable),
percentage)) +
  geom_point() +
  theme_classic() +
  theme(axis.text.x = element_blank()) +
  xlab("Variable Number") +
  ylab("Percentage of Total Importance")
print(p.imp)
```

根据数据集的描述，前 12 个变量给出了音乐的各种音色，接下来的 78 个变量是来自前 12 个变量的协方差矩阵的唯一成分。有意思的是，在最重要的变量中，前三个都是音色，而不是来自于协方差。例如，如果后 78 个变量收集起来的代价很高或者很困难，我们或许会考虑只使用前 12 个变量可能会是什么样的性能。下面的模型测试了使用简单浅层模型的方法。

```
mtest <- h2o.deeplearning(
  x = colnames(d)[2:13],
  y = "V1",
  training_frame= h2omsd.train,
  validation_frame = h2omsd.test,
  activation = "RectifierWithDropout",
  hidden = c(50),
  epochs = 100,
  input_dropout_ratio = 0,
  hidden_dropout_ratios = c(0),
  score_training_samples = 0,
  score_validation_samples = 0,
  diagnostics = TRUE,
  export_weights_and_biases = TRUE,
  variable_importances = TRUE
)

mtest
```

```
    H2ORegressionModel: deeplearning
    Model ID: DeepLearning_model_R_1452082402089_15
    Status of Neuron Layers: predicting V1, regression, gaussian
    distribution, Quadratic loss, 701 weights/biases, 13.6 KB, 27,
398,762
    training samples, mini-batch size 1
      layer units             type dropout       l1       l2 mean_rate
    1   1    12              Input  0.00 %
    2   2    50 RectifierDropout  0.00 % 0.000000 0.000000  0.003773
    3   3     1             Linear          0.000000 0.000000  0.000985
      rate_RMS momentum mean_weight weight_RMS mean_bias bias_RMS
    1
    2 0.007925 0.000000    0.004197   0.504967 -0.679546 0.965184
    3 0.000926 0.000000   -0.106522   0.286619 -1.400430 0.000000


    H2ORegressionMetrics: deeplearning
    ** Reported on training data. **
    Description: Metrics reported on full training frame

    MSE:  82
    R2 :  0.24
    Mean Residual Deviance :  82


    H2ORegressionMetrics: deeplearning
    ** Reported on validation data. **
    Description: Metrics reported on temporary (load-balanced)
validation frame

    MSE:  83
    R2 :  0.22
    Mean Residual Deviance :  83
```

结果表明，R^2对于训练数据仅为0.24，对于测试数据仅为0.2，这依然比得上

使用所有变量的线性回归，但是远低于在完整的预测变量集合上使用神经网络获得的 0.32 或 0.35。即使很多变量只有相当小的重要性，把它们加起来就有了显著的差异。

5.6 小结

在本章中，我们深入地描述了什么是深度神经网络，特别是如何使用它们来训练预测模型。尽管深度前馈神经网络看起来相当复杂，它们也可以分解成一系列的层，每一层都是相当简单的，还带有一个输入集和一个输出集以及把两者映射到一起的权重和偏差。

我们已经看到了使用深度学习在预测性能上的可能的提升。在用例中，使用线性回归只解释了测试数据中 23%的变化。然而，使用深度前馈神经网络，我们能够解释歌曲发布年份中 35%的变化。尽管远谈不上完美，这依然是在回归性能之上的戏剧性的提升。而且低性能更多可能是与缺乏解释逐年差异的数据有关而不是由于模型本身（换句话说，没有更多/更好的预测变量即使用最好的模型也不太可能解释 99%的变化）。接下来，本书的第 6 章将要介绍如何调节和优化模型，包括如何解决某些常见的挑战，比如缺失数据或者低劣的模型准确度/性能。

第 6 章
调节和优化模型

在本章中，我们将讨论一些用来调节模型的方法。这会涉及解决缺失数据的方法。虽然我们已经使用过没有缺失数据的案例数据集，但在真实数据中有缺失数据是司空见惯的。我们还将讨论当模型的性能不佳时能做什么，包括如何寻找并优化模型超参数的详细的检查。

这一章将包括下列主题。

- 处理缺失数据
- 低准确度模型的解决方案

在本章中，我们将使用两个新包：用于作图的 gridExtra 包以及用于拟合广义可加模型的 mgcv 包。我们应该将这些新包添加到 `checkpoint.R` 文件中，获取这个文件，用来为余下显示的代码创建 R 环境。我们使用下列代码，建立 R 并且初始化 H2O 集群。

```
source("checkpoint.R")
options(width = 70, digits = 2)

cl <- h2o.init(
  max_mem_size = "12G",
```

```
  nthreads = 4)
```

6.1 处理缺失数据

在处理真实世界的应用时，通常我们需要应付缺失数据。H2O 包括了一个使用均值、中位数或者众数插补变量的函数，而且其他的一些分组变量也可以选择用来做插补。为了检查如何使用这种方式插补缺失数据，我们将使用关于花朵的短小的 Iris 数据集。为此，我们将把品种 setosa 的花瓣宽度（petal width）和长度值设置为缺失，然后插补它们的值。

```
## setup iris data with some missing
d <- as.data.table(iris)
d[Species == "setosa", c("Petal.Width", "Petal.Length") := .(NA, NA)]

h2o.dmiss <- as.h2o(d, destination_frame="iris_missing")
h2o.dmeanimp <- as.h2o(d, destination_frame="iris_missing_imp")
```

首先，我们将做简单的均值插补，这需要每次处理一列。

```
## mean imputation
missing.cols <- colnames(h2o.dmiss)[apply(d, 2, anyNA)]

for (v in missing.cols) {
  h2o.dmeanimp <- h2o.impute(h2o.dmeanimp, column = v)
}
```

估算全体非缺失值均值的一个问题是，如果存在任何系统性的差异，这些差异将会被遗漏。而且，如果我们能从任何非缺失的数据中得到关于缺失数据的更好的预测，差异也将被错过。

我们能使用一个简单的预测模型代替简单的均值插补。下列代码建立了一个随

机森林模型，来预测每个缺失的列，使用了所有的缺省值。如果随机森林用时太长，也能使用 glm 模型。

```
## random forest imputation
d.imputed <- d

## prediction model
for (v in missing.cols) {
  tmp.m <- h2o.randomForest(
    x = setdiff(colnames(h2o.dmiss), v),
    y = v,
    training_frame = h2o.dmiss)
  yhat <- as.data.frame(h2o.predict(tmp.m, newdata = h2o.dmiss
))
  d.imputed[[v]] <- ifelse(is.na(d.imputed[[v]]), yhat$predict,
d.imputed[[v]])
}
```

为了比较不同的方法，我们可以创建一个花瓣长度对花瓣宽度的散点图，散点的颜色和形状由花瓣的种类来决定。这张图有三个子图。上方的图是初始数据。中间的图是使用了均值插补的数据。下方的图是使用随机森林插补的数据。下列代码创建了如图 6-1 所示的图形。

```
grid.arrange(
  ggplot(iris, aes(Petal.Length, Petal.Width,
    color = Species, shape = Species)) +
    geom_point() +
    theme_classic() +
    ggtitle("Original Data"),
  ggplot(as.data.frame(h2o.dmeanimp), aes(Petal.Length, Petal.
Width,
    color = Species, shape = Species)) +
    geom_point() +
    theme_classic() +
    ggtitle("Mean Imputed Data"),
```

```
ggplot(d.imputed, aes(Petal.Length, Petal.Width,
  color = Species, shape = Species)) +
  geom_point() +
  theme_classic() +
  ggtitle("Random Forest Imputed Data"),
ncol = 1)
```

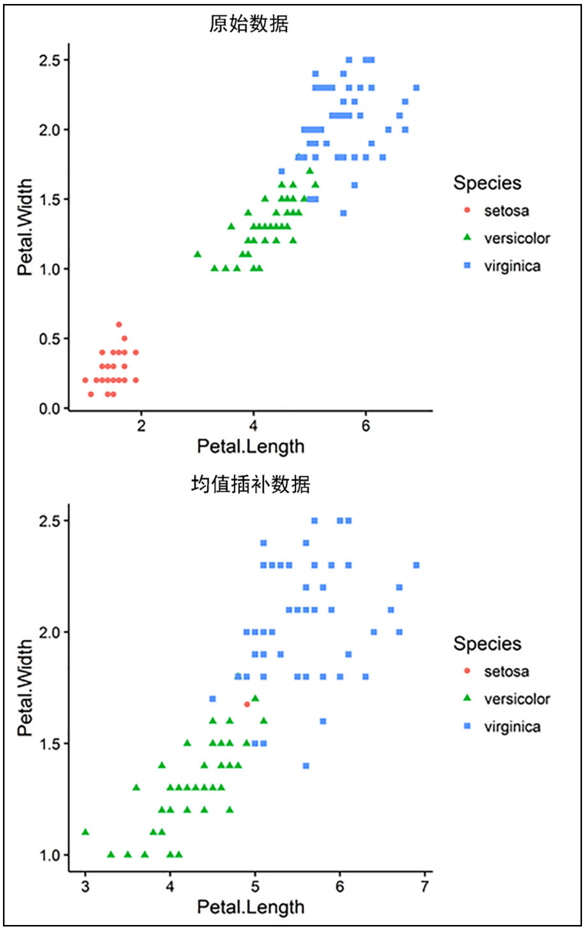


图 6-1

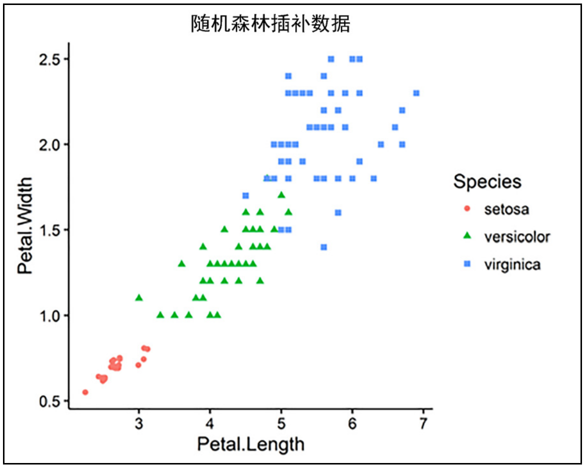


图 6-1（续）

在这个例子中，均值插补创建的异常值相当地偏离了现实，如果有需要，应该生成更高级的预测模型。在统计推断中，多重插补要优先于单一插补（无论何种方法），因为后者未能考虑不确定性。就是说，当插补缺失数据时，因为要明确那些值到底是什么，存在着某种程度的不确定性。然而，在大多数深度学习的用例中，数据集实在太大而且计算时间要求太多，无法做到创建有不同插补值的多重数据集，而且在每一个数据集上训练模型并且汇总结果。于是，这些更简单的方法（比如均值插补或者使用其他的预测模型）是常用的。

6.2 低准确度模型的解决方案

最具有挑战性的，也是潜在优化模型方面最重要的是为超参数选择取值。理论上，我们想要选择最好的组合，虽然我们不可能真的找到全局最优，但本节中的技术能够帮忙找到更好的超参数取值。更好的超参数通常能提升模型的准确度。

然而有时候，一个模型有坏的准确度要归咎于缺乏好的预测所要求的变量，或者是因为没有足够的数据去支持训练一个足够复杂的模型来准确地预测或分类数据。这些情况也会要求获取一些额外的变量（特征），用以预测和（或）其他情况。

本书不能帮助读者收集更多的数据，但是它能介绍调节和优化超参数的方法。接下来我们将处理这一点。

6.2.1 网格搜索

关于调节超参数的更多的信息，参见 Bengio, Y. (2012)，特别是第 5 章第 3 节，讨论了各种超参数的选择和刻画。除了手动试错法，另外两种可以提升超参数的方法是网格搜索和随机搜索。在网格搜索中，我们可以对超参数指定几个值，然后尝试所有可能的组合。这种方法或许是最容易看到的。在 R 中，我们能使用 `expand.grid()` 函数来创建所有可能的变量组合。

```
expand.grid(
  layers = c(1, 2, 4),
  epochs = c(50, 100),
  l1 = c(.001, .01, .05))

   layers epochs    l1
1       1     50 0.001
2       2     50 0.001
3       4     50 0.001
4       1    100 0.001
5       2    100 0.001
6       4    100 0.001
7       1     50 0.010
8       2     50 0.010
9       4     50 0.010
10      1    100 0.010
11      2    100 0.010
12      4    100 0.010
13      1     50 0.050
14      2     50 0.050
15      4     50 0.050
16      1    100 0.050
```

```
17        2     100 0.050
18        4     100 0.050
```

当这几个参数只有少数几个取值的时候，网格搜索是优异的。然而，尽管这是一种评估不同参数取值的全面的方法，当对某些或许多个参数有很多取值的时候，它就迅速变成不可行的了。例如，即使 8 个参数中的每一个都只有两个取值，就有了 $2^8=256$ 种组合，这样很快在计算上就不切实际了。而且，如果参数和模型性能之间没有相互作用，或者这种相互作用相对于主要效应很小，那么网格搜索就是无效的方法，因为许多参数值被重复了，这样尽管尝试了许多组合，也只是抽样了取值的一个小集合。

6.2.2 随机搜索

一种替代的方法是通过随机抽样进行搜索。与其指定所有的值去尝试并去创建所有的组合，不如为参数随机地抽样取值、拟合模型、存储结果并重复这个过程。为了得到非常大的样本规模，也会有计算上的需求，但这确实能直接指定有多少个我们愿意运行的模型。

对随机抽样来说，所有需要被指定的就是要随机抽样的值或者要随机抽出的分布。通常，我们也要设置一些限制。例如，尽管一个模型在理论上有任何整数个层，抽样使用的是一些合理的数（例如 1～10）而不是抽样一到一百万的整数。

为了做随机抽样，我们将写出一个函数，它取一个随机数种子然后随机地抽取一些超参数，存储抽取的参数，运行模型并返回结果。即使做一次随机搜索来尝试找到更好的值，我们也不是从所有可能的超参数中来抽样。许多超参数保持在我们指定的固定值或者它们的缺省值上。

对某些参数来说，我们可以采取一些处理来指定如何来随机抽样取值。例如，当为了正则化而使用丢弃时，对于输入变量通常有一个较少数量的丢弃（一般大约 20%），对于隐藏神经元有一个较大数量的丢弃（一般大约 50%）。选择恰当的分布

可以允许我们把这种先验信息编码到随机搜索中。下列代码画出了两个 beta 分布的密度，结果如图 6-2 所示。通过在这些分布中抽样，我们能确保搜索，尽管是随机的，重点在于输入变量小比例的丢弃上，而且在 0～0.50 的范围内对隐藏神经元来说，有一种趋势是从更接近 0.50 的值当中过抽样（oversample）。

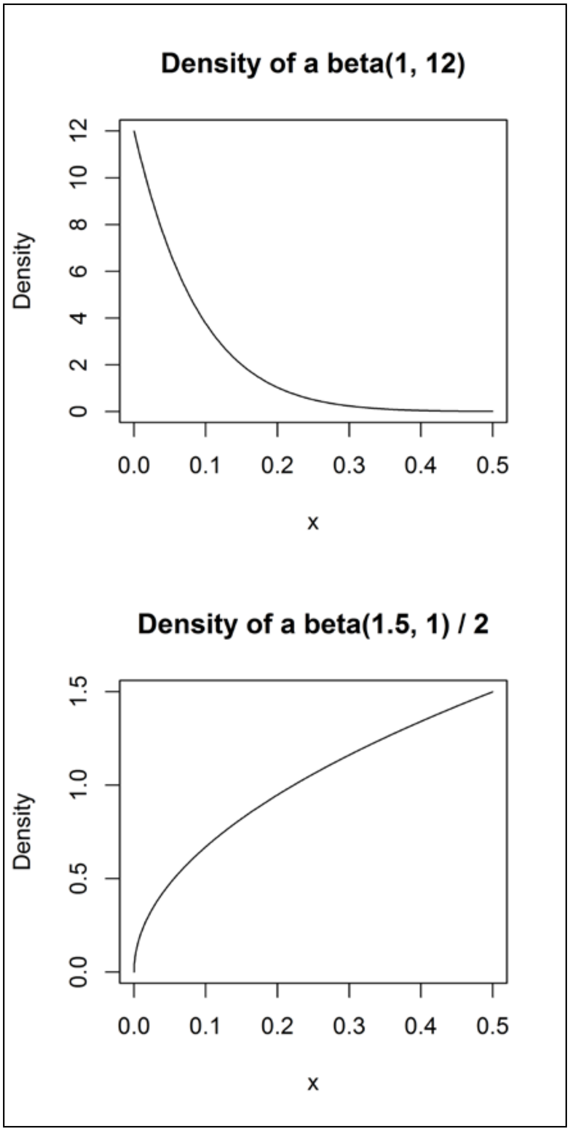


图 6-2

```
par(mfrow = c(2, 1))
plot(
  seq(0, .5, by = .001),
  dbeta(seq(0, .5, by = .001), 1, 12),
  type = "l", xlab = "x", ylab = "Density",
  main = "Density of a beta(1, 12)")

plot(
  seq(0, 1, by = .001)/2,
  dbeta(seq(0, 1, by = .001), 1.5, 1),
  type = "l", xlab = "x", ylab = "Density",
  main = "Density of a beta(1.5, 1) / 2")
```

现在我们能写这个函数了，把它叫作 `run()`。它所需要的只是一个随机种子，用来使参数选择的过程可重复。我们可以指定一个参数的名字，尽管基于种子有一个缺省值而且有一个可选的（逻辑）参数 `run`，用来控制模型是否在运行。如果你想检查所抽样的超参数的值，这个参数是有帮助的。

我们从 1～5 来抽样深度或者层数，从 20～600 来抽样每一层中的神经元个数。默认每个抽出的值有相等的概率。`runif()`函数从均匀分布指定的范围中抽样，而且我们已经见到了 beta 分布，使用 `rbeta()`函数从 beta 分布中抽样。

我们还要随机抽取的两个新参数是 `rho` 和 `epsilon`。使用这两个参数是因为预期手动地指定学习率和动量（momentum），我们使用（H2O 默认这样做）ADADELTA 算法（Zeiler, M. D. (2012)）来自动地调整学习率。ADADELTA 只有两个超参数需要指定：`rho` 和 `epsilon`。ADADELTA 起作用在某种程度上是通过检查之前的梯度，而不是存贮之前所有的梯度，用到了一个加权的累积平均。`rho` 参数用于在当前迭代之前加权梯度，`1 - rho` 用于加权当前迭代的梯度。如果 `rho=1`，那么不使用当前的梯度而完全基于之前的梯度；如果 `rho=0`，不使用之前的梯度而完全基于当前的梯度。通常，使用的 `rho` 值在.9～.999 之间。

`epsilon` 参数是一个小的常数，在取之前平方梯度的均方根时，它被加到上

面（目的是避免这些均方根恰好变成零）而且通常它是一个非常小的数。进一步的说明由给出 ADADELTA 的论文提供（Zeiler, M. D. (2012)）。

```
run <- function(seed, name = paste0("m_", seed), run = TRUE) {
  set.seed(seed)

  p <- list(
    Name = name,
    seed = seed,
    depth = sample(1:5, 1),
    l1 = runif(1, 0, .01),
    l2 = runif(1, 0, .01),
    input_dropout = rbeta(1, 1, 12),
    rho = runif(1, .9, .999),
    epsilon = runif(1, 1e-10, 1e-4))

  p$neurons <- sample(20:600, p$depth, TRUE)
  p$hidden_dropout <- rbeta(p$depth, 1.5, 1)/2

  if (run) {
  model <- h2o.deeplearning(
    x = colnames(use.train.x),
    y = "Outcome",
    training_frame = h2oactivity.train,
    activation = "RectifierWithDropout",
    hidden = p$neurons,
    epochs = 100,
    loss = "CrossEntropy",
    input_dropout_ratio = p$input_dropout,
    hidden_dropout_ratios = p$hidden_dropout,
    l1 = p$l1,
    l2 = p$l2,
    rho = p$rho,
    epsilon = p$epsilon,
    export_weights_and_biases = TRUE,
    model_id = p$Name
```

```
  )

  ## performance on training data
  p$MSE <- h2o.mse(model)
  p$R2 <- h2o.r2(model)
  p$Logloss <- h2o.logloss(model)
  p$CM <- h2o.confusionMatrix(model)

  ## performance on testing data
  perf <- h2o.performance(model, h2oactivity.test)
  p$T.MSE <- h2o.mse(perf)
  p$T.R2 <- h2o.r2(perf)
  p$T.Logloss <- h2o.logloss(perf)
  p$T.CM <- h2o.confusionMatrix(perf)

  } else {
    model <- NULL
  }

  return(list(
    Params = p,
    Model = model))
}
```

在运行模型之前，我们需要载入数据，用于这个例子的数据是活动数据。

```
use.train.x <- read.table("UCI HAR Dataset/train/X_train.txt")
use.test.x <- read.table("UCI HAR Dataset/test/X_test.txt")

use.train.y <- read.table("UCI HAR Dataset/train/y_train.txt")
[[1]]
use.test.y <- read.table("UCI HAR Dataset/test/y_test.txt")[[1
]]

use.train <- cbind(use.train.x, Outcome = factor(use.train.y))
use.test <- cbind(use.test.x, Outcome = factor(use.test.y))
```

```
use.labels <- read.table("UCI HAR Dataset/activity_labels.txt")

h2oactivity.train <- as.h2o(
  use.train,
  destination_frame = "h2oactivitytrain")

h2oactivity.test <- as.h2o(
  use.test,
  destination_frame = "h2oactivitytest")
```

为了使得参数可重复，我们指定一个随机种子的列表，循环这个列表来运行模型。

```
use.seeds <- c(403L, 10L, 329737957L, -753102721L, 1148078598L,
-1945176688L,
-1395587021L, -1662228527L, 367521152L, 217718878L, 1370247081L,
571790939L, -2065569174L, 1584125708L, 1987682639L, 818264581L,
1748945084L, 264331666L, 1408989837L, 2010310855L, 1080941998L,
1107560456L, -1697965045L, 1540094185L, 1807685560L, 2015326310L,
-1685044991L, 1348376467L, -1013192638L, -757809164L, 1815878135L,
-1183855123L, -91578748L, -1942404950L, -846262763L, -497569105L,
-1489909578L, 1992656608L, -778110429L, -313088703L, -758818768L,
-696909234L, 673359545L, 1084007115L, -1140731014L, -877493636L,
-1319881025L, 3030933L, -154241108L, -1831664254L)
```

通过循环随机数种子，我们可以简单地运行模型（尽管会花费一点时间）。

```
model.res <- lapply(use.seeds, run)
```

一旦模型完成，我们能创建一个数据集，使用下面的代码，针对不同的参数画出均方误差（MSE）。结果显示如图 6-3 所示。

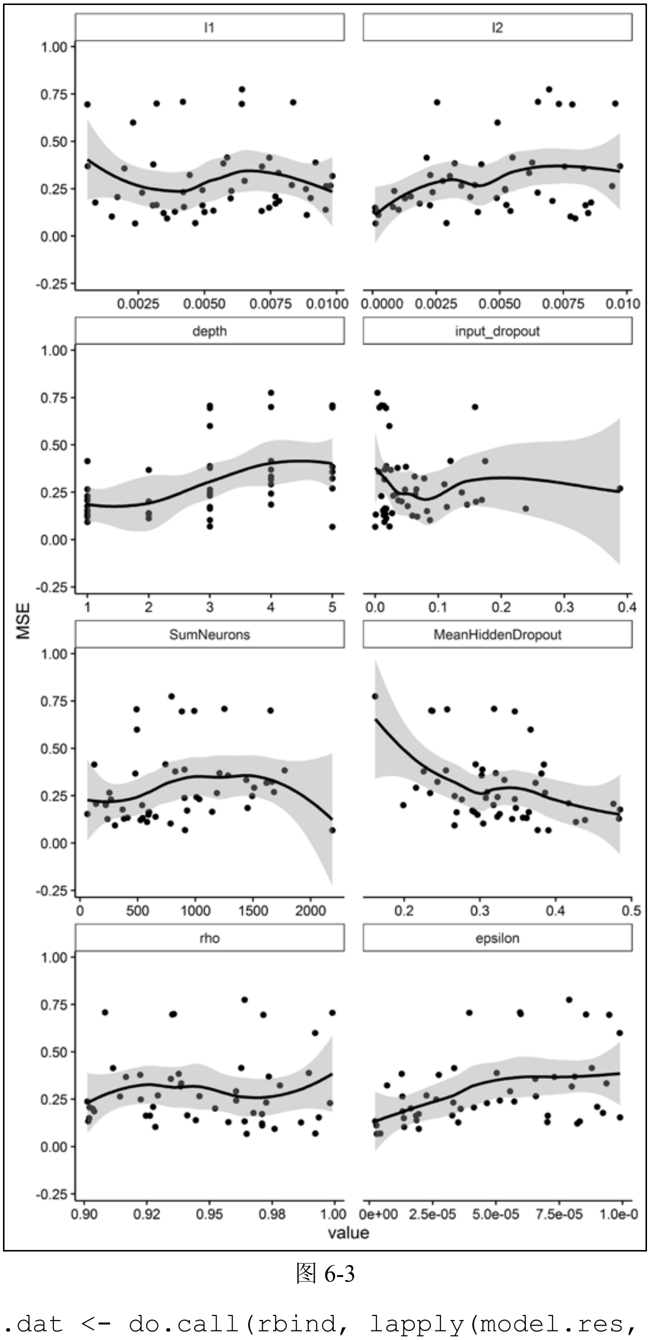


图 6-3

```
model.res.dat <- do.call(rbind, lapply(model.res, function(x)
with(x$Params,
```

```
      data.frame(l1 = l1, l2 = l2,
                 depth = depth, input_dropout = input_dropout,
                 SumNeurons = sum(neurons),
                 MeanHiddenDropout = mean(hidden_dropout),
                 rho = rho, epsilon = epsilon, MSE = T.MSE))))


p.perf <- ggplot(melt(model.res.dat, id.vars = c("MSE")), aes(
value, MSE)) +
      geom_point() +
      stat_smooth(color = "black") +
      facet_wrap(~ variable, scales = "free_x", ncol = 2) +
      theme_classic()
print(p.perf)
```

除了观察参数之间的一元联系和模型误差，使用多元模型同时取不同的参数来考虑也是有帮助的。

为了拟合这个多元模型（以及允许一些非线性），采用来自 mgcv 包的 `gam()` 函数，我们使用一般可加模型（generalized additive model）。在模型深度和隐藏神经元的总个数之间，我们明确地假设一种相互作用，通过使用 `te()` 函数把这两部分包括在一个张量扩展当中，我们能捕捉这种相互作用。我们使用 `s()` 函数对其余的项给定一元光滑。这里的指定并不是非常重要的。要决定应该选择什么样的参数取值，关键是以某种方式建模超参数和模型性能之间的联系。

```
m.gam <- gam(MSE ~ s(l1, k = 4) +
              s(l2, k = 4) +
              s(input_dropout) +
              s(rho, k = 4) +
              s(epsilon, k = 4) +
              s(MeanHiddenDropout, k = 4) +
              te(depth, SumNeurons, k = 4),
            data = model.res.dat)
```

现在我们能可视化结果。使用下列代码，前 6 个一元的项我们画在一张图中，如图 6-4 所示。常数项没有显示，所以这些取值不是直接的 MSE 估计，但关键是找到每一个超参数的最低点。

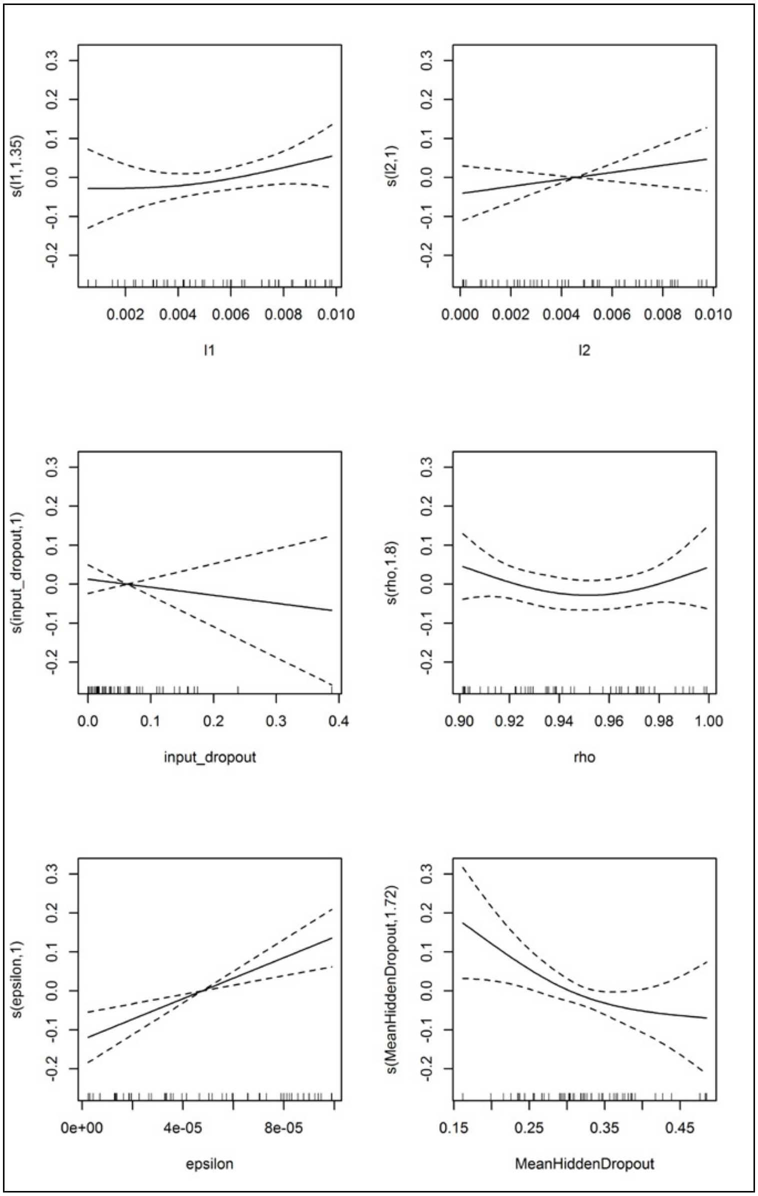


图 6-4

```
par(mfrow = c(3, 2))
for (i in 1:6) {
  plot(m.gam, select = i)
}
```

最后，我们用下面代码画出交换作用项。

```
plot(m.gam, select = 7)
```

结果如图 6-5 所示。这是一张等值线图，显示了每一个变量在 *x* 轴和 *y* 轴的交互作用。真实的 MSE 没有显示出来，但是在线上做了标记。因为交互作用，我们使用不同的预测变量组合可能得到相同的 MSE 估计。一般来说，似乎是层数越多，要获得可比较的性能就需要越多的神经元。

基于这些图，在下列代码中，我们选择超参数并指定一个“优化的”模型。

```
model.optimized <- h2o.deeplearning(
    x = colnames(use.train.x),
    y = "Outcome",
    training_frame = h2oactivity.train,
    activation = "RectifierWithDropout",
    hidden = c(300, 300, 300),
    epochs = 100,
    loss = "CrossEntropy",
    input_dropout_ratio = .08,
    hidden_dropout_ratios = c(.50, .50, .50),
    l1 = .002,
    l2 = 0,
    rho = .95,
    epsilon = 1e-10,
    export_weights_and_biases = TRUE,
    model_id = "optimized_model"
)
```

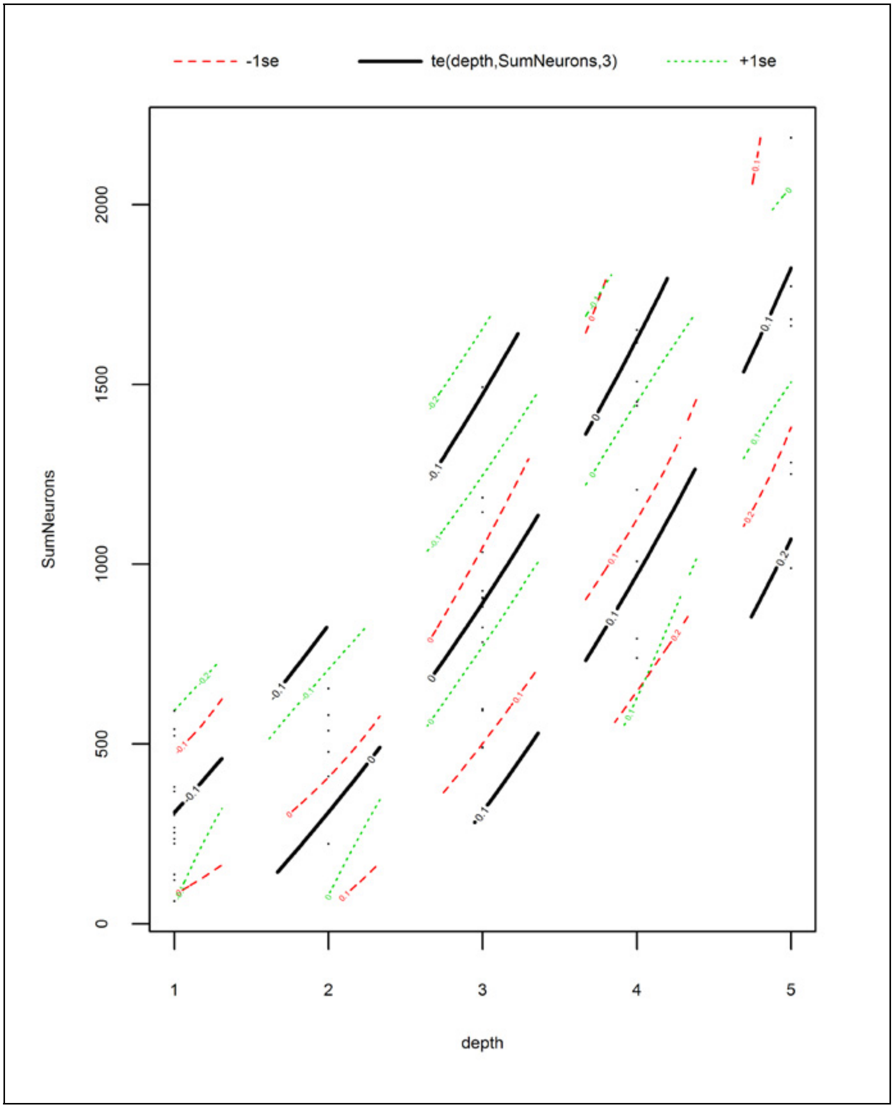


图 6-5

在模型训练之后，使用 h2o.performance()函数并将优化的模型和测试数据作为参数传递给它，我们能在验证数据中估计模型的性能。

```
H2OMultinomialMetrics: deeplearning
```

```
Test Set Metrics:
=====================

MSE: (Extract with 'h2o.mse') 0.053
R^2: (Extract with 'h2o.r2') 0.98
Logloss: (Extract with 'h2o.logloss') 0.18
Confusion Matrix: Extract with 'h2o.confusionMatrix(<model>,
<data>)')
==============================================================
        X1   X2   X3   X4    X5   X6   Error        Rate
1      491    0    5    0     0    0   0.010      5 / 496
2       12  457    1    0     1    0   0.030     14 / 471
3       32   47  341    0     0    0   0.188     79 / 420
4        0    2    0  434    55    0   0.116     57 / 491
5        0    0    0   38   494    0   0.071     38 / 532
6        0    0    0    0    15  522   0.028     15 / 537
Totals 535  506  347  472   565  522   0.071 208 / 2,947

Hit Ratio Table: Extract with 'h2o.hit_ratio_table(<model>,
<data>)'
==============================================================
Top-6 Hit Ratios:
  k hit_ratio
1 1  0.929420
2 2  0.993892
3 3  0.998643
4 4  0.999661
5 5  1.000000
6 6  1.000000
```

最后，针对来自随机搜索的单个最佳模型，我们能比较我们模型的性能。使用优化了的参数，我们能在测试数据中得到 0.053 的 MSE，比起随机搜索中得到的单个最佳模型减少了差不多 21%。

```
model.res.dat[which.min(model.res.dat$MSE), ]
```

```
        l1      l2 depth input_dropout SumNeurons MeanHiddenDropout
18 0.0024 0.00011     5         1e-04       2186              0.39
    rho epsilon   MSE
18 0.96   3e-06 0.067
```

在本节中，我们介绍了如何搜索各种超参数、使用图形和一些建模方法以及如何尝试选择更好的超参数。更正式的优化超参数也是可能的，比如对参数的贝叶斯优化使用 Spearmint 库，我们可以在线得到 https://github.com/HIPS/Spearmint。尽管这些微调和优化的例子是对深度预测模型显示的，它们也可以运用到预测和异常检测中。

6.3 小结

利用第 4 章和第 5 章中的技术，可以建立并使用深度自动编码器来学习数据中的特征、识别离群点和异常值以及使用深度前馈神经网络来预测新的结果或分类数据，如图像、语音或其他的数据。尽管只是一个入门，希望来自本书的思想和代码能够使我们开启深度学习，解决真实世界的实际问题。

深度学习和人工智能是非常活跃的研究领域。新的工具和技术层出不穷，本书只是对深度学习中的一些标准和常用的模型提供了一个介绍。现在就是学习这个领域的激动人心的时刻，笔者希望本书能有助于开启大家的旅程。

参考文献

以下是本书中所有引文的参考文献。

- Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. (2013). A Public Domain Dataset for Human Activity Recognition Using Smartphones. 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013. Bruges (Belgium), 24-26 April 2013.

- Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. In Neural Networks: Tricks of the Trade (pp. 437-478). Springer Berlin Heidelberg. (Also on the arXiv: http://arxiv.org/pdf/1206.5533.pdf)

- Bengio, Y., Courville, A., and Vincent, P. (2013). Representation Learning: A Review and New Perspectives. Pattern Analysis and Machine Intelligence, IEEE Transactions, 35(8), 1798-1828.

- Bergmeir, C., and Benítez, J. M. (2012). Neural Networks in R Using the Stuttgart Neural Network Simulator: RSNNS. Journal of Statistical Software, 46(7), 1-26.

- Bishop, C. M. (2006). Pattern Recognition and Machine Learning, Springer.

- Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout Networks. arXiv preprint arXiv:1302.4389.

- Hastie, T., Tibshirani, R., and Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition. Springer.

- Hinton, G. E., Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural Computation, 18 (7), 1527-1554.

- Kuhn, M. (2008). Building Predictive Models in R Using the caret Package. Journal of Statistical Software, 28 (5), 1-26. Appendix[152]

- Kuhn, M. and Johnson, K. (2013). Applied Predictive Modeling. New York: Springer.

- Lichman, M. (2013). UCI Machine Learning Repository (http://archive.ics.uci.edu/ml). Irvine, CA: University of California, School of Information and Computer Science.

- Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT press.

- Nair, V., and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10) (pp. 807-814).

- Riedmiller, M., and Braun, H. (1993). A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In Neural Networks, 1993, IEEE International Conference.

- Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. Neural Networks, 61, 85-117.

- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

- Venables, W. N. and Ripley, B. D. (2002). Modern Applied Statistics with S-Plus. Fourth Edition. Springer.

- Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P. A. (2008, July). Extracting and Composing Robust Features with Denoising Autoencoders. In Proceedings of the 25th International Conference on Machine Learning (pp. 1096-1103). ACM.

- Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. arXiv preprint arXiv:1212.5701.

深度学习精要（基于 R 语言）

R Deep Learning Essentials

深度学习是机器学习的一个分支，其基础是一组试图使用模型架构建立高水平抽象模型的算法。本书结合 R 语言介绍深度学习软件包 H2O，帮助读者理解深度学习的概念。本书从在 R 中设置可获取的重要深度学习包开始，接着转向建立神经网络、预测和深度预测等模型，所有这些模型都由实际案例的辅助来实现。成功安装了 H2O 软件包后，你将学习预测算法。随后本书会解释诸如过拟合数据、异常数据以及深度预测模型等概念。最后，本书会介绍设计调参和优化模型的概念。

本书的目标读者

本书适合那些胸怀大志的数据科学家，他们精通 R 语言数据科学概念，并希望可以使用 R 中的包进一步探索深度学习范式。读者需要对 R 语言具备基础的理解，并熟悉统计算法和机器学习技术。

通过阅读本书，你将能够

- 建立 R 包 H2O 训练深度学习模型；
- 理解深度学习模型背后的核心概念；
- 使用自动编码器识别异常数据或离群点；
- 使用深度神经网络自动化地预测或分类数据；
- 使用正则化建立泛化模型，避免数据的过拟合。

异步社区 www.epubit.com.cn
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn

ISBN 978-7-115-46415-6

分类建议：计算机/深度学习/R
人民邮电出版社网址：www.ptpress.com.cn